Princeton University

Operations Research and
Financial Engineering

Jet Engines' Supply Chain Optimization: Graphical Utility and One-Agent Inventory Policy against Uncertainties

Author
Woramanot Yomjinda

Supervisor
Professor Warren B. Powell

April 25, 2018

Introduction

In this research under the advisory of Professor Warren Powell, I have been introduced to a multi-agent simulator for optimizing the supply chain for jet engines. The simulator created by Professor Powell's graduate student at the CASTLE Labs, Dionysios Kalogerias, is at an early stage where it can produce a file that feeds to PilotView to create a graphical utility to show flows dynamically.

The main challenge of this Independent Work is to develop policies to make the system more robust to randomness. To this end, Dionysios Kalegerias have greatly supported this independent work on the algorithmic side. The main study I have done which will be in this report are on the study of the dynamics of the network, testing of the robustness of new policies, and the results from running the simulations.

The simulator itself starts off at a rudimentary One-agent only state. I have improved upon the simulator through several means. The successful attempts include code-debugging against time and ordering error, introducing parallelization which greatly reduces run-time during multiple simulations and is useful for when randomness is introduced to the system. Most importantly, the inventory policy and Hiccup system have been successfully introduced, albeit very complicated to understand. There are further improvements that can be made including Multi-agent layering and Bias-learning model which I will explain in detail before the closing of this report.

Supply Chain Simulation
Part I: The Five Components

First, it is important to realize that at a current stage, the Supply Chain itself is deterministic i.e. there is no randomness, no learning process. But it is also equally important to derive the five components: State Variables, Decision Variables, New Information, Transition Function, and Objective Function. So, the model will have a strong ground work for when the uncertainties and learning process are added to the system.

State Variables:

We define the state variable to be

$S_t = (D_t, U_t, P_t, I_t, O_t)_j$

For each node j and a horizon H, node j has a single parent node which it feeds out parts to, and possibly multiple or zero children nodes that supply its input parts

$D_t \in R^H$ is a vector of Parent's Upstream Demand for node j's product from period t-1

For each of j's children node i with travelling time c,

$U_t^i \in R^{H-c}$ are vectors of Children's Downstream Supply to feed to node j from period t-1

$P_t^i \in R^c$ are vectors of Children's Projected Shipments to feed to node j from period t-1

$I_t^i \in R^{\#i}$ is node j's current input inventory from each children i

$O_t \in R$ is node j's current output inventory

Supply Chain Simulation
Part I: The Five Components

Decision Variables:

We define the decision variable to be

$x_t^j = x^{\pi,j}(S_t) = (X_{t+1}^j, (D_{t+1}^j)_i)$ where i represents j's children node

During each period, each node j puts states S_t through a optimization policy that gives $X_{t+1} \in R^{H-c+1}$ a production plan for node j where c is travel time to parent node

At the same time, for each of j's children node i,
$(D_{t+1})_i \in R^H$ is node j's Upstream Demand which will become a Demand information for node i's State variable $(S_t)_i$.

New Information:

Since this is a multi-agent supply chain, the information observed by each node after time t is the information sent by its parent and children node at time t i.e.

$(W_{t+1})_j = (X_{t+1}, (D_{t+1})_i)_{j's\ children}$ & $(X_{t+1}, (D_{t+1})_i)_{j's\ parent}, \forall j$

The updating equation then take places as followed

Supply Chain Simulation
Part I: The Five Components

Transition function:

Our function takes S_t and W_{t+1} and produces

$S_{t+1} = (D_{t+1}, U_{t+1}, P_{t+1}, I_{t+1}, O_{t+1})_j$

Where, if we call j's parent k

$D_{t+1} = \left(D^k_{t+1}\right)_j$ i.e. j's Parent Upstream demand from period t

For each Children i,
$P^i_{t+1} = Append(P^i_t[1:H], \min(O^i_t + X^i_{t+1}[0], D^i_{t+1}[0]))$

We append at the end either the children's remaining inventory plus product produced by children node i at time t up to the value that node j demanded last period. At the same time, Product $P^i_t[0]$ arrived last period so it is cut out

$U^i_{t+1} = X^i_{t+1}[1: H - C + 1]$ or a new production plan from children node

$I^i_{t+1} = P^i_{t+1}[0] + I^i_t - X^j_{t+1}[0] * Part\ i\ neededPerProd$:
Product $P^i_{t+1}[0]$ arrived this period and we updated the part lost from last period production.

$O_{t+1} = \max(O_t + X_{t+1}[0] - D_{t+1}[0], 0)$ We ship out whatever inventory+new production we have up to the Parent's node demand.

Supply Chain Simulation
Part I: The Five Components

Objective Function:

As a general idea, each node is Myopic (only cares about period t) and wants to minimize the function below over the Horizon i.e. t to t+H

$$\min \sum_{n=0}^{H}\{\theta[n] * \ Unmet[n] + KO * O_t[n] + KPro * X_{t+1}[n] + Ipsum\left(KI^i * I_t^i[n]\right) + Ipsum\left(KPur^i * \left(D_{t+1}^j\right)_i[n]\right)\}$$

According to constraints,

$$If\ n = 0, RI_{previous} = RI_{current}: else, RI_{previous} = RI_{Vars}[n-1]$$
$$If\ n = 0, RO_{previous} = RO_{current}: else, RO_{Previous} = RO_{Vars}[n-1]$$
$$RI_{Vars}^i[n] - RI_{previous}^i + Q^i * X_{t+1}[n] - \left(D_{t+1}^j\right)_i[n] - P_t^i[n] = 0, \forall i$$
$$RO_{Vars}[n] - RO_{previous} - X_{t+1}[n] + D_t[n] - Unmet[n] = 0$$
$$Unmet[n] - D_t[n] \leq 0$$

Where our decision variables are $(X_{t+1}^j, \left(D_{t+1}^j\right)_i)$

And

Unmet[n] is the projected Parent node's Unmet demand

KO is cost of keeping output inventory per unit of product

KPro is cost of producing per unit output

KI^i is cost of keeping input inventory per unit of part from children node i

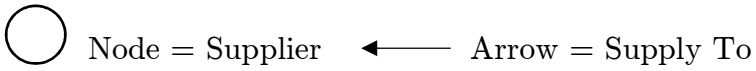
$KPur^i$ is cost of purchase per unit of part from children node i

Which we solved using either Gurobi or CPLEX

Supply Chain Simulation
Part II: The Simulator

Now that we gain an understanding of the Supply Chain optimization theoretically, we will learn more about the flow and mechanism side of the whole Supply Chain Simulator. In this section, I have presented a rigorous Graphic guide which will lead the reader through the whole process behind the working of the One-Agent Supply Chain Simulation which is the current state of our program.

Part I: Supply Chain Tree



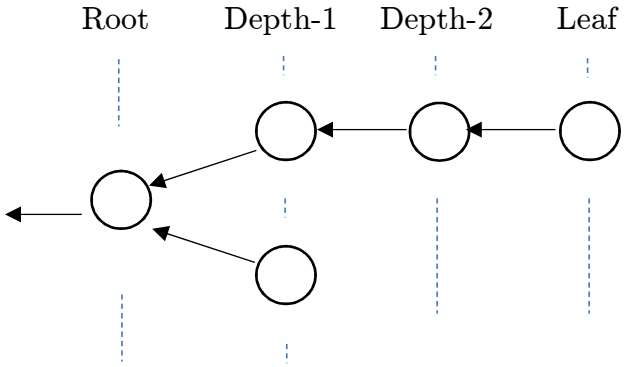


Figure 1: Valid Tree of Depth 4

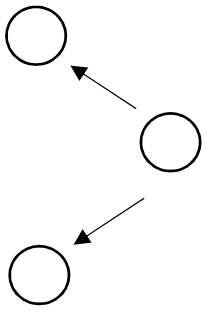


Figure 2: Invalid Tree

The tree can have as many depths as we want, at the cost of simulation speed.

Every node represents different supplier within different depth as suggested by the graphics, Due to the nature of the simulation, each supplier (child) from depth D only supplies to another supplier (parent) in depth D+1. In other words, each supplier can receive supplies from multiple suppliers but cannot supply to multiple suppliers. Supplier also cannot jump above or below over one tier (although we can certainly permit that without compromising simulation speed or complexity at all). Finally, the jet-engine supplier (root) supplies to a customer with a constant demand.

The data are generated in the above way with depth = 4

Supply Chain Simulation
Part II: The Simulator

Part II: Supply Chain Dynamics

The whole process is tunable with variables H: Horizon and T: Time. We discretized time into T periods with each period equaling one day. Within each day, a simulation process runs through every supplier in the tree. The process is as followed.

Update Shipment List—Produce Parts & Update Private Values—Update Global Values by Private—Repeat

The order of supplier we put through each stage does not matter since each supplier has their own dictionary which contains both public (pre-updated) and private (post-updated) values. We, of course, will introduce this dictionary of each supplier which is the core of our simulation below.

Supply Chain Simulation
Part II: The Simulator

Part III: The Supplier Dictionary: & The importance of each attribute

Global Attributes: Horizon (This is a tunable parameter)

Stationary Attributes:

Label: 8576 if Root

Production Cap:

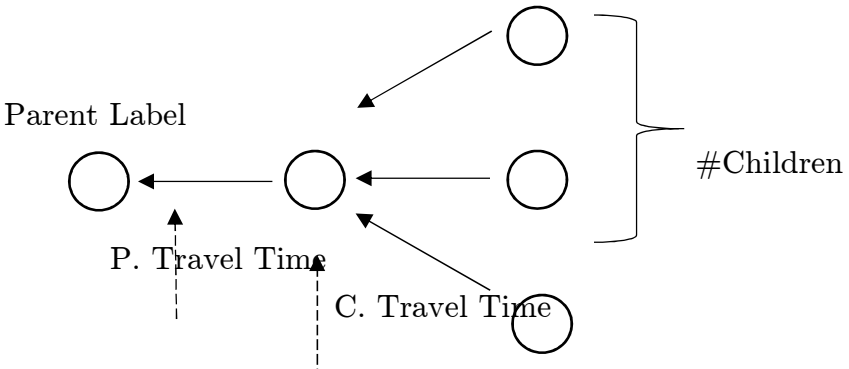
Value: 1 if Lead

Latitude & Longitude: For PilotView and Travel Time Calculation

Depth: 0 if Root

Fixed Correlated Attributes:

Children Labels



Product Demands: Upstream part demand per produced part

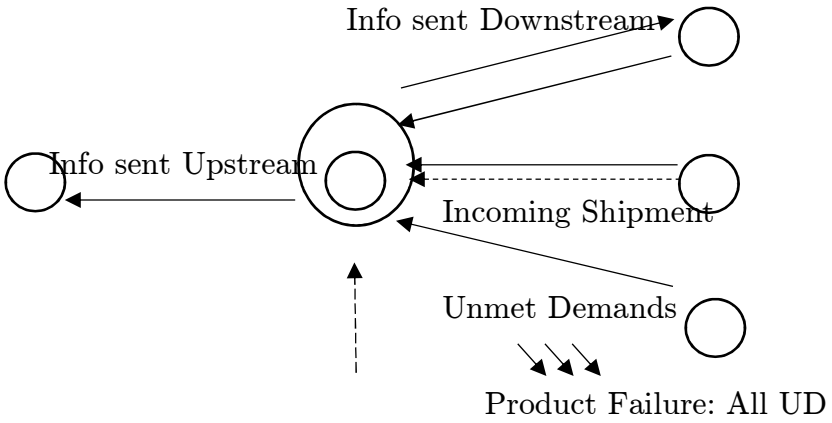
KI, KO, Kpro, Kpur

The Ks are Input Cost, Stock Cost, Production Cost, and Purchase Cost respectively

Supply Chain Simulation
Part II: The Simulator

Dynamic Correlated Attributes:

Public (PRE): Data from (t-1) which is key to supplier's decision making in period (t)



Note! Info sent Downstream is actually "Upstream" info for the children and info sent Upstream is "Downstream" info for the parent

Private (POST): Data that we store during period (t) so it does not influence the updating process until the current stage is completed. These are

Info sent Upstream(t), Info sent Downstream(t), and Incoming Shipment(t)

Supply Chain Simulation
Part II: The Simulator

Part IV: The Day-to-Day Updating Code

As stated before, the process is.

Update Shipment List—Produce Parts & Update Private Values—Update Global Values by Private—Repeat

The code below comes from file SupplyChainWork, all data are initialized through dataPrep to fill in the dictionaries.

Update Shipment List:

```
# Update shipment list for EACH supplier
for ID, value in SupplierDict.items():
    if SupplierDict[ID].NumberOfChildren != 0:
        # Iterate in a COPY of the current ShipmentList
        for shipment in SupplierDict[ID].ShipmentList_PRE[:]:
            # Update shipment in ShipmentList of current Supplier
            shipment.LocalShipmentUpdate(SupplierDict[ID])
    SupplierDict[ID].ShipmentList_POST = cp.deepcopy(SupplierDict[ID].ShipmentList_PRE)
```

What happens here is that
1.) we choose all supplier with at least one children
2.) Each shipment has timer attached, we move the timer down by 1 day
3.) if the timer reaches -1, we add the inventory to the Input Inventory
4.) Finally, we update shipment(t) to be equal to the updated shipment (t-1).

Supply Chain Simulation
Part II: The Simulator

Produce Parts & Update Private Values:

```
    # Produce Parts (and "PRIVATELY" update attributes) for EACH Supplier
for ID, value in SupplierDict.items(): # This should be able to be performed in parallel
    #print('Day', t, '/ Updating Suppler ID:', int(ID))
    # Get label of parent to current Supplier
    TheParent = SupplierDict[ID].ParentLabel
    # Get the plans from all children to current Supplier
    tempSpec = dict()
    # ALSO: Compute TOTAL input inventory for current Supplier (with no children)
    tempTotalInv = 0
    if SupplierDict[ID].NumberOfChildren != 0:
        for child in SupplierDict[ID].ChildrenLabels:
            tempSpec[child] = SupplierDict[child].DownStream_Info_PRE
            tempTotalInv += SupplierDict[ID].InputInventory[child]


    # Produce parts for today and update Supplier
    #++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
    SupplierDict[ID].ProduceParts(SupplierDict[TheParent] if TheParent != -1 else -1,
                DataFromChildren = tempSpec,
                DataFromParent = SupplierDict[TheParent].UpStream_Info_PRE[ID] if TheParent != -1 else RootPlan[t : t + H])
    #++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
```

For each supplier, the following happens
1.) if have at least one child, we gather all Downstream information(t-1) first, calling it tempspec
2.) if current node is not root, we also gather Upstream information from parent as well.
3.) We project inventories using shipment list(t-1) and children travel time, then run the projected inventories, Upstream/ Downstream data along with Dictionary attributes through a Black Box called ProduceParts (In SupplierClasses.py)
4.) This Black Box uses LookaheadMIP solved through Gurobi to formulate optimal production plans, update Up/Downstream Information(t), Input/Output inventory and Unmet demands
5.) If the demand is met, then the shipment is made with timer = Parent travel time.
Update Global Values by Private:

```
# Update PRE variables with POST variables
for ID, value in SupplierDict.items():
    SupplierDict[ID].ShipmentList_PRE = cp.deepcopy(SupplierDict[ID].ShipmentList_POST)
    SupplierDict[ID].DownStream_Info_PRE = cp.deepcopy(SupplierDict[ID].DownStream_Info_POST)
    SupplierDict[ID].UpStream_Info_PRE = cp.deepcopy(SupplierDict[ID].UpStream_Info_POST)
```

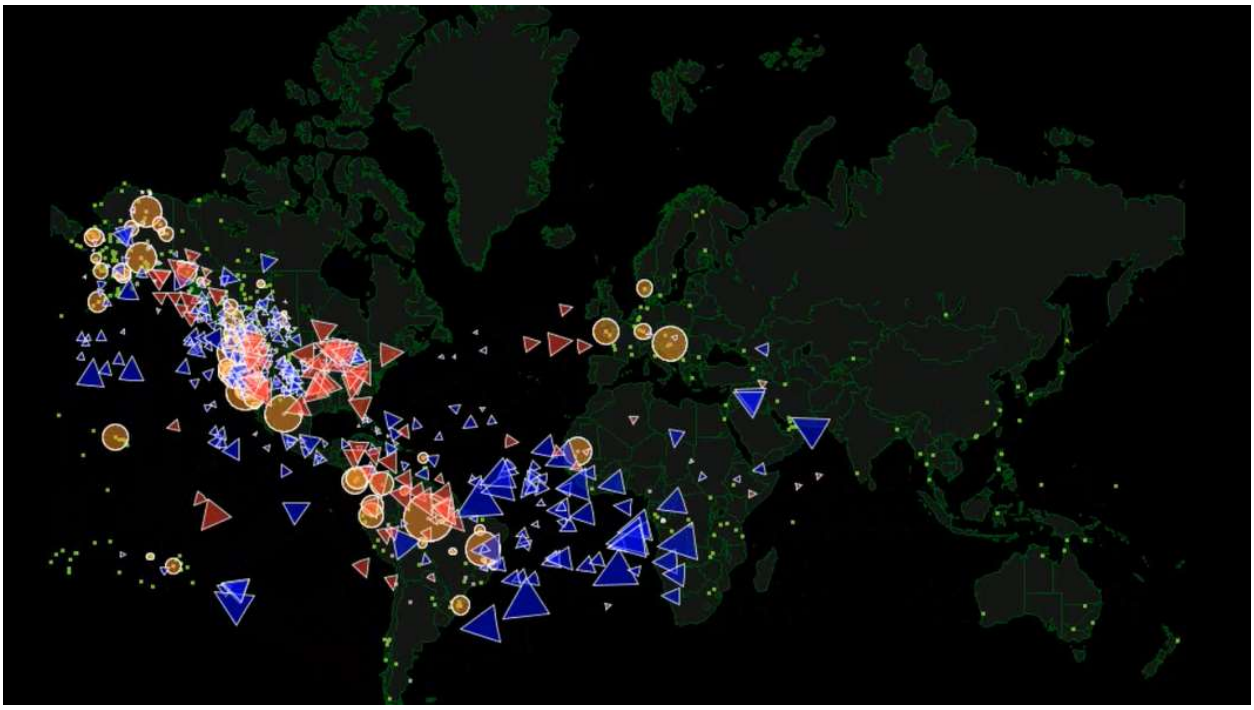
We prepare for period (t+1) by replacing all Up/DownStream Info(t-1) with Up/DownStream Info(t) (which is a t-1 for period t+1) and all ShipmentList(t-1) with ShipmentList(t).

Supply Chain Simulation
Part II: The Simulator

Part V: What the Simulator Does

As we can see from the updating process, the simulator itself updates the Shipment list, the inventories, and the Unmet Demands throughout the period t= 0,…,T.

By recording these three values in relation to Label & time t, we can graphically represent their dynamics/ flows using PilotView program.



The Blue arrow represents PartFlow (Value of arriving shipment at t).
The Red arrow represents Unmet Demand (Value of parts needed but not arrived at t).
The Orange Circle represents Output Inventory at a given time t.
By utilizing this graphic video, we will be able to introduce hiccup to the system which introduces problem to the system. And, when the problem is introduced, we will be able to introduce policies to counteract the problem and optimize the policies against the stochastic hiccups.

Supply Chain Simulation
Part III: Simulations and Hiccups

Through Part I: The Five Components and Part II: The Simulator, I hope that the reader is able to gain the insight into the working of the Supply Chain Simulation both theoretically and practically. This Part III: Simulations and Hiccups intends to illustrate and analyzes the findings resulted from using Supply Chain Simulator with PilotView throughout this Semester of Independent Work

Simulations:

We start off by assuming no Hiccups. We set horizon H = 13 and total time T = 100
The pictures below shows each stage of the simulation

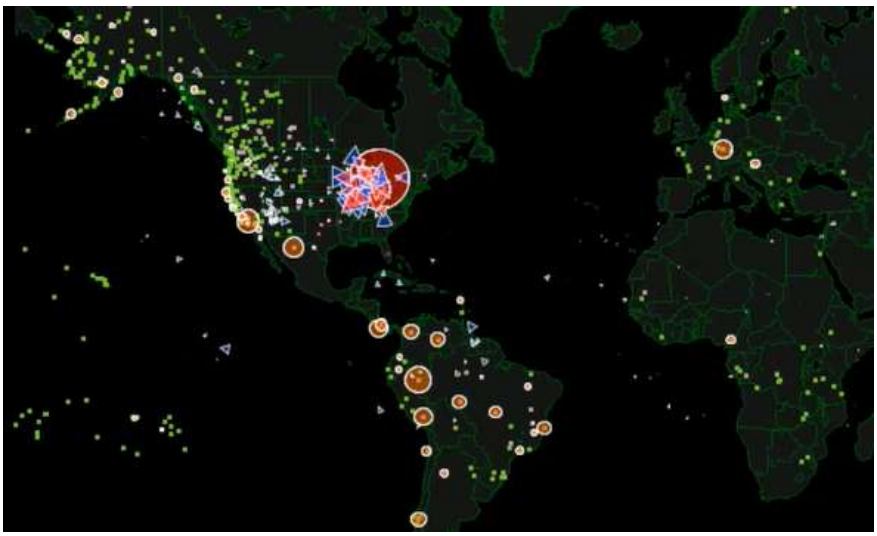
Initial State: The root node (Jet Engine producer) is not able to produce enough Jet Engine due to lack of input parts. This kick starts the whole system because the root starts the whole Upstream Demand chain.

Stabilizing State: The root produces as much as it can while the Demand chain start to function to feed into the layer by layer until reaching the root

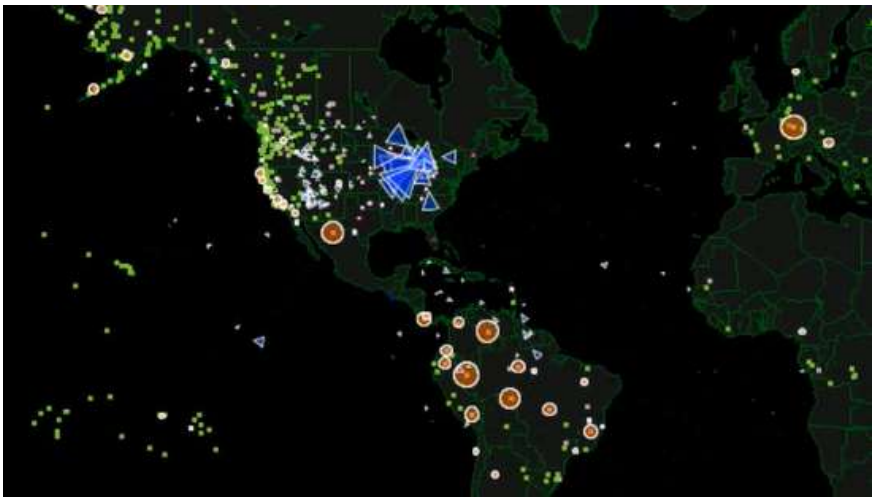
Supply Chain Simulation
Part III: Simulations and Hiccups

Stable State: This state can be explained by a cyclical behavior of the root node i.e. the root satisfies the Unmet demand period by period as shown below

Period 1: The root does not satisfy Jet Engine demand immediately. This is done to balance the output inventory storage cost.



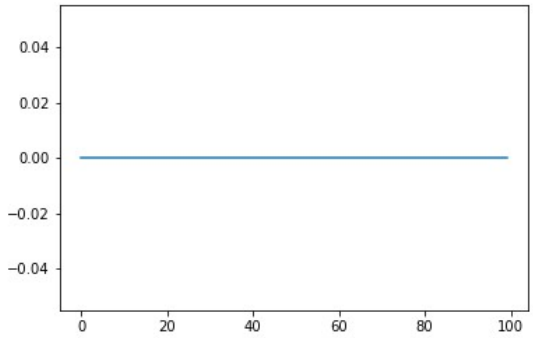
Period 2: As we can see, the root then is able to produce Jet Engine to satisfy the demand while taking in large amount of Input inventory.



The two periods take turn in a cyclical manner in a stabilized state.

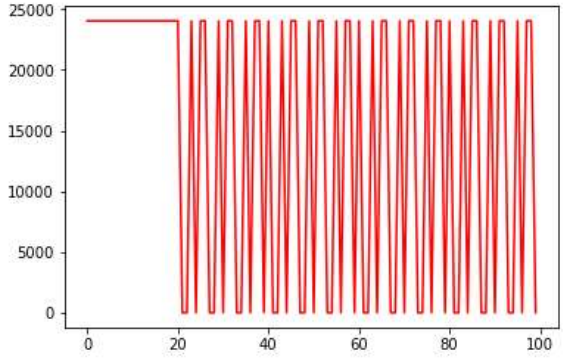
Supply Chain Simulation
Part III: Simulations and Hiccups

The take away from Simulation with no Hiccup is best shown in the graph below,
No Hiccup: Output Inventory



The Jet Engine producer never want to store output inventory
This results in the following Unmet Demand graph

No Hiccup: UnMet Demands



Now, we want to introduce Hiccup to the system and observe the effect of the hiccup. Interestingly, we first try to put Hiccup at time t = 33 to t = 40 (1-week hiccup) and there is almost zero difference (at least not observable by naked eyes) to the PilotView output. This is because of the high input inventory of each node including the Jet Engine producer which allows them to survive through a short period of Hiccup, up to almost 14 days.

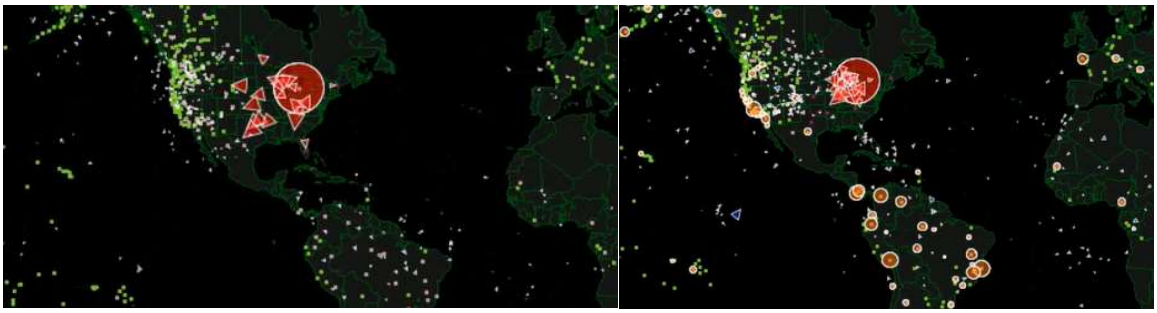
This means that if we want the Hiccup to have any serious consequences, we will have to make sure that the Hiccup lasts longer than 14 days.

Supply Chain Simulation
Part III: Simulations and Hiccups

Hiccups:

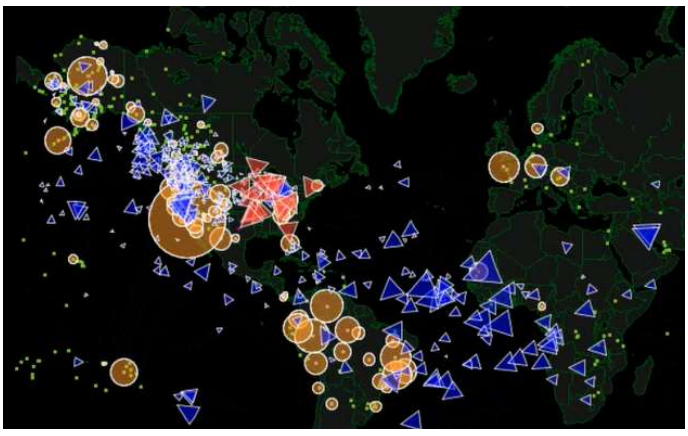
We let time horizon H = 13 and time T = 100. Now, we introduce Hiccup at time t = 33 to t = 66 by setting KPRO = 10000000 (cost of production at node j that we want Hiccup to happen) and observes the effect of the Hiccups

The initial and stabilizing period before the Hiccup hits look the same:



Now, when the Hiccup hits, the cyclical behavior lasts for about 10-12 days before the Jet Engine producer is no longer able to produce anymore Jet Engine due to a lack of input material

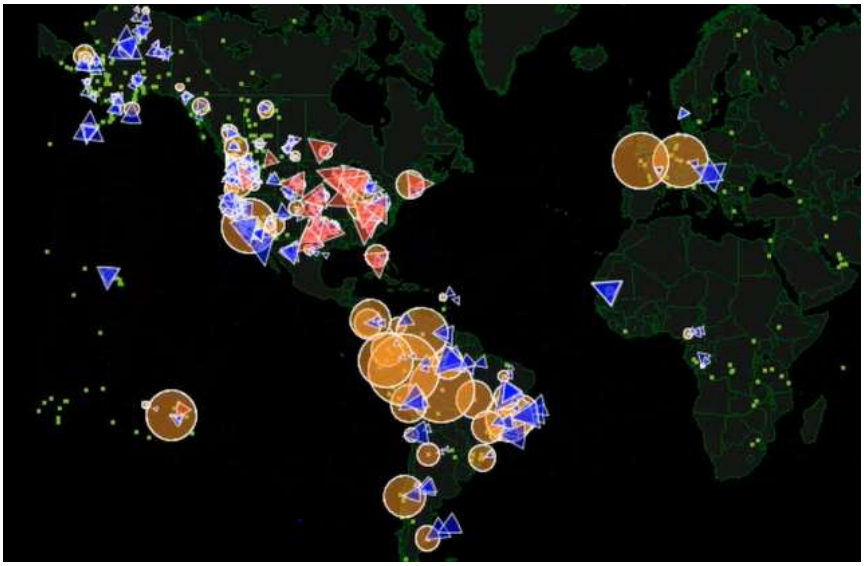
Initial Period after Hiccup: (Note that we have done some scaling to make it easier to see Output Inventory and flows of less valuable parts here) The whole system is continuing in cyclical manner as usual



Supply Chain Simulation
Part III: Simulations and Hiccups

After day 43: Almost the whole system shut down despite many nodes having output inventories. This is because the root can no longer produces Jet Engine so the demand is now zero, kicking the whole system haywire.

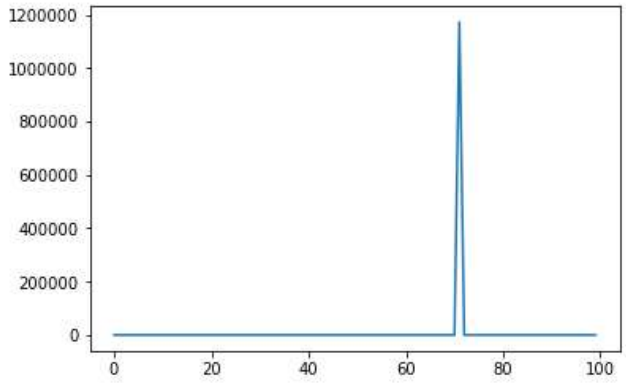
After day 67: The whole system returns to the production schedule extremely quickly and in an even more rushed manner, using up most of stockpiled output inventory.



Supply Chain Simulation
Part III: Simulations and Hiccups

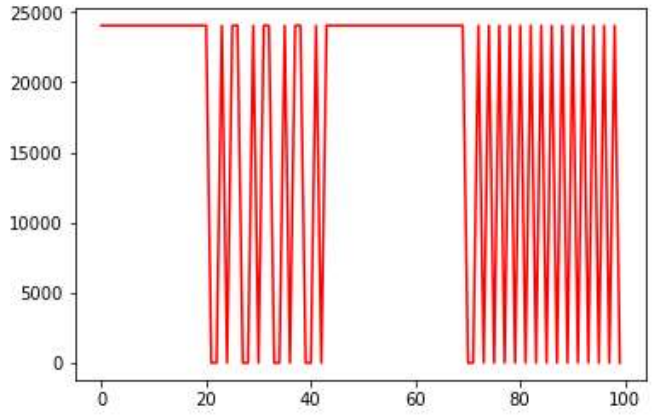
Again, the take away from Simulation with Hiccup(33,66) is best shown in the graph below,

Hiccup(33,66): Output Inventory



As we can see, the Jet Engine producer is willing to use up all input inventories to create output inventories to satisfy its parent's demand because it finally anticipates incoming input material that is now out of Hiccup

Hiccup(33,66): Unmet Demand



From the graph above, the period from t = 43 to t = 66 is when the whole system starts the complete shutdown until the shutdown ends. After the shutdown is over, the Jet Engine producer speeds up the cyclical behavior greatly compared to the period before shutdown.

Supply Chain Simulation
Part IV: Inventory Policy

To combat this shut down, the most expensive but also the only one available to One-Agent Supply Chain Simulator is of course, the inventory policy i.e. forcing each node to have at least X days of output inventory ready for shipping whenever a Hiccup happens.

Due to the nature of Gurobi and CPLEX optimization, we refer to the objective function

$$\min \sum_{n=1}^{H} \{\theta[n] * Unmet[n] + KO * O_t[n] + KPro * X_{t+1}[n] + Ipsum\left(KI^i * I_t^i[n]\right) + Ipsum\left(KPur^i * \left(D_{t+1}^j\right)_i[n]\right)\}$$

Now, for the above objective to be feasible, however, we cannot straight up add a constraint on the output X_{t+1}
Instead, we accomplished this by using two parameters shown in the code below where number 30 and 18 lies

```
Inv_floor = np.multiply(Q[child]*(1-np.divide(RO_Current, 30))*18,
                        np.divide((np.divide(np.sum(D),H-2-ChildrenTrTimes[child])),H-2-ChildrenTrTimes[child]))
UPD_Values[child].append(max(int(UPD_Vars[t][child].varValue),int(Inv_floor)))
```

The first number 30 refers to the convergence end goal. You can think of this as a shape-shifting graph from 0 to 30 that tries to balance a ball (current inventory) to about the middle value (15)
The second number 18 refers to the speed of convergence but it also has an impact on the final stabilizing value. In this case, it shifts stabilizing value from 15 to 16. It is important to have this value since slower stabilization might not create enough inventory before the Hiccup hits. However, having too high a number of this can also create an illusion that a policy is "safe" as will be show in the next page

Supply Chain Simulation
Part IV: Inventory Policy

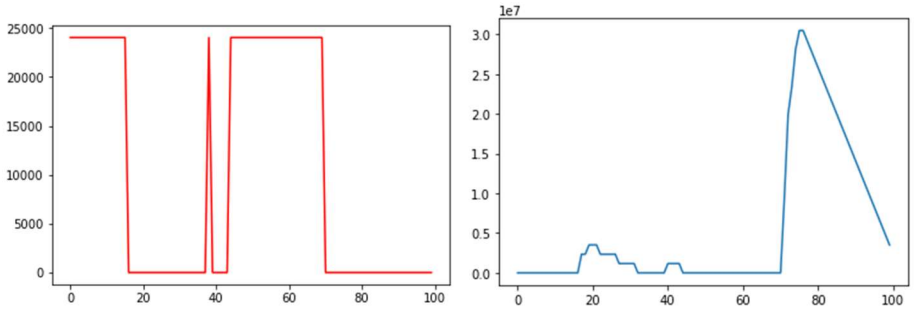
Starting from this page, we will look extensively at the result of implementing Inventory policy. It is first important to note that while PilotView provides amazing imagery, it can be quite difficult to analyze since we do not know the exact size of current inventory as shown below

.

Inventory Policy: The parts are flowing to the Jet Engine Producer and the system does not shut down even at t=57. Yet, we do not know from the size of the circle how many output inventory is left for the root node.

Therefore, we will look instead at Output Inventory graph of various inventory policy. It is important to note that we can imply Unmet Demand Graph easily from the Output inventory graph since the period where the Output Inventory flats out at 0 during Hiccup is a period where the system shutdown and Unmet Demand reigns

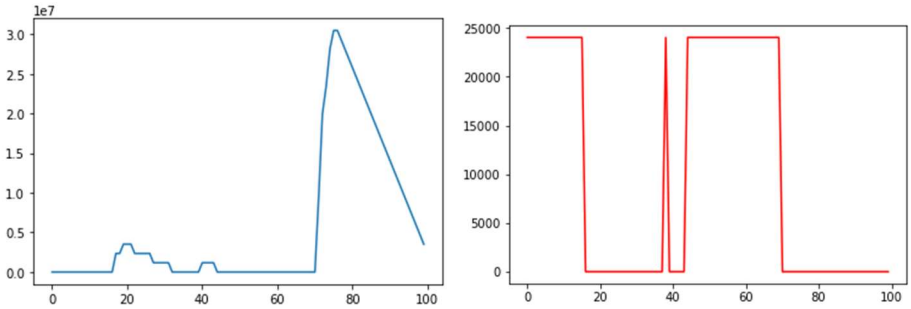
For instance, Hiccup(33,66) of the graph below Output Inv. implies Unmet Demand



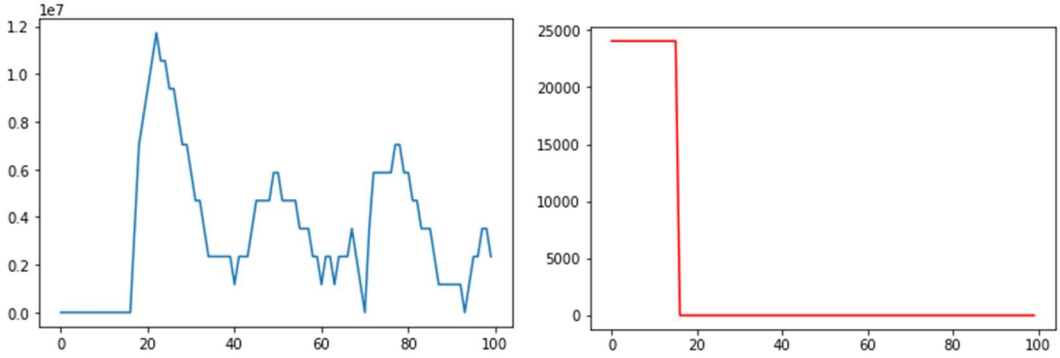
Supply Chain Simulation
Part IV: Inventory Policy

Starting with the unsuccessful policies already shown in the last page
We represent policies by Policies(x,y) where x is the end cap and y is the speed of balancing. The Hiccup hits by default at (33,66) unless stated otherwise

Policies(12,3): With balancing force too slow, the system is doomed to shut down as Jet Engine producer only has 3 output inventory ready



Policies(12,9): This policy gives illusion of being perfect. This s because the balancing force is extremely fast enough at the beginning that it still have some output inventory left when hit by Hiccup at 33. The inventory of this policy itself only lasts 9 days. The hiccup lasts 33 days but with initial 12 inventories and enough input inventory to sustain for about 12 days, the Jet Engine producer survive by a straw at the end of day 66 and is able to recover greatly from day 67 onwards.

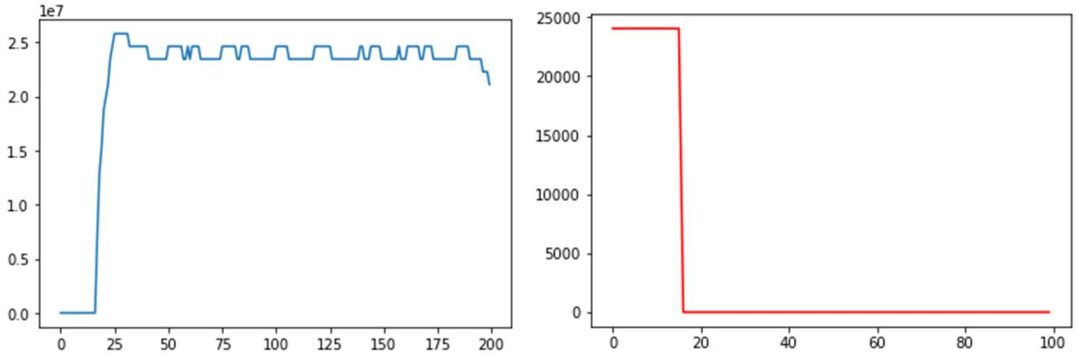


Of course, we do not want our policy to simply luck out by starting off quickly. Thus, we decide to observe over a longer period of time i.e. let the root node stabilizes output inventory first so there is no more luck out.

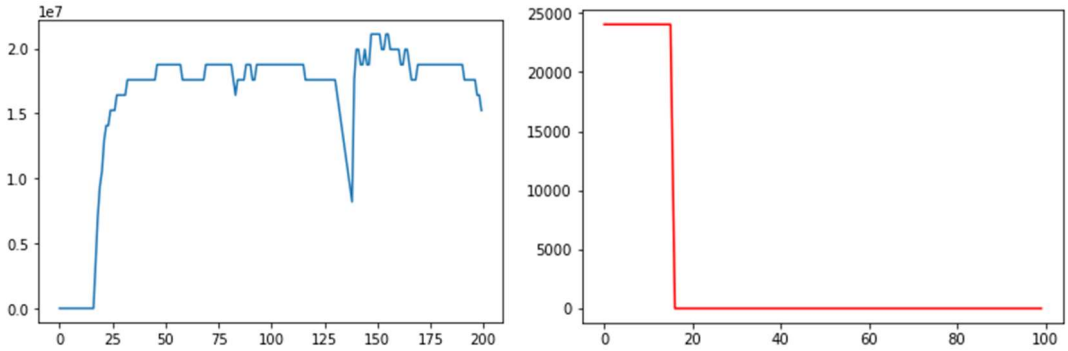
Supply Chain Simulation
Part IV: Inventory Policy

We then set T = 200 and let the Hiccup starts at 100-133. The horizon is the same at H = 13 and represent policies by Policies(x,y) where x is the end cap and y is the speed of balancing.

Policies(50,25): We start by making sure that the inventory policy actually works. Because we stabilizes at 25, the Jet Engine producer has abundance of input inventory to sustain over 33 days of hiccup and almost no need for the output inventory created at all.



Policies(30,18): This is where the graphs illustrate well that the inventory policy works by forcing the Jet Engine producer to create Output inventory worth 18 days which we can see that those are used from period 125 to 133 when the root node used up all input inventory to satisfies day by day demand.



Supply Chain Simulation
Part V: Multi-Agent and Bias-Learning

Inventory policy allows us to combat uncertainties greatly, but also at a great cost
Not only is the storage cost expensive, this policy forces every node to keep output inventory worth large amount of money, some even more than the root node as shown on the right picture.

This is partially my fault for not being able to create a multi-agent system. While Multi-agent system would require intensive layering of algorithm of this program, what it does is that it allows for the node to alternatively purchase the part that normally only produced by children K from another children J. This, however, means that children J must be able to produce part produced by children K in the first place. Even so, this might mean that J is selling the part at a higher price until K can no longer produce the part due to hiccup.

This is quite complicate because node J will have to split the input inventory into producing either product which adds multiple layer of complication. However, for the node that do not produce 2 products or more this only changes the objective function to

$$\min \sum_{n=1}^{H}\{\theta[n] * \ Unmet[n] + KO * O_t[n] + KPro * X_{t+1}[n] + argmin_i(Ipsum\left(KI^i * I_t^i[n]\right) + Ipsum\left(KPur^i * \left(D_{t+1}^j\right)_i[n]\right))\}$$

Where $argmin_i$ choose from all alternative vendors that produces the "same" input parts.
As we can induct from the algorithm above, having alternative vendors, while greatly complicate the algorithm will be able to solve Hiccup problem by itself as long as the Hiccup does not happen at choke point (where the part is only produce by one vendor) greatly alleviating the need for inventory policy and open rooms for even more interesting policy

Supply Chain Simulation
Part V: Multi-Agent and Bias-Learning

Theoretically, albeit with possible unforeseen difficulties in implementation, we can derive the alteration in the Five components of Multi-Agent Alternate-Vendors Simulation below

State Variables: $S_t = (D_t, U_t, P_t, I_t, O_t)_j$ For each node j with parent possibly above 1

$D_t^M \in R^H$ is a vector of Parent's M Upstream Demand for node j's product

$O_t^M \in R$ is node j's current output inventory for product that feeds parent M

Decision Variables: $x_t^j = x^{\pi,j}(S_t) = \left(\left(X_{t+1}^j\right)_M, \left(D_{t+1}^j\right)_i\right)$

where i represents j's children node and M represents j's parent nodes

New Information: $(W_{t+1})_j = (X_{t+1}, (D_{t+1})_i)_{j's\ children}$ & $(X_{t+1}, (D_{t+1})_i)_{j's\ parent}, \forall j$

But in this case, there are now multiple parents

Transition function:

For each parent M of node j:

$D_{t+1}^M = (D_{t+1}^M)_j$ i.e. j's Parent Upstream demand

$O_{t+1}^M = \max({O_t}^M + {X_{t+1}}^M[0] - D_{t+1}^M[0], 0)$ We ship out whatever inventory + new production we have up to the Parent's node M's demand.

Objective Function:

$\min \sum_{n=1}^{H} \{\theta^M[n] * Unmet^M[n] + KO^M * O_t^M[n] + KPro^M * X_{t+1}^M[n] +$

$argmin_i (Ipsum\left(KI^i * I_t^i[n]\right) + Ipsum\left(KPur^i * \left(D_{t+1}^j\right)_i[n]\right))\}$

Where for each part needed, we can choose to buy from the cheapest vendor and for parts produced, we produce to the most profitable vendor. The problem that needs to be fixed, however, is that it is unrealistic that we can ignore certain vendors until Hiccup hits and it might create an unforeseen problem where the system shut itself down because one non-crucial product is more profitable for children to produce than the other product that can only be produce by this children.

Supply Chain Simulation
Part V: Multi-Agent and Bias-Learning

Even if we ignore Multi-agent/ alternative vendors system altogether, the bias learning should still be kept in mind. For Bias-learning, the key idea is that instead of or in addition to having inventory policy, each node can now ask for more than it needs for instance, node 1 may ask node 2 for 30 parts while only need 20 parts when the shipping date comes.

This can help combat Hiccup problem greatly but is also dangerous at the same time. Because in reality, there should be a bias-learning process, the more times the node 1 lies to node 2, the less likely node 2 believes that node 1 actually needs that many parts and will produce less in response.

Currently, our model does not take into account the bias-learning at all so it is always fine to ask a lot then only need a few. If implemented, Bias-learning that starts collecting data from period 1 up to t may greatly reduce the effectiveness of the inventory policy but may also increases its efficiency at the same time by making sure there is no over-ordering of any parts.