**Abstract:**

The following document outlines the projects to be undertaken in Summer 2019.

**N Strikes Problem**

This deals with the evaluation of the current shipment policy for Cisco products across the board. Currently Cisco ships a product to a customer. Either the customer is satisfied and he accepts the product or he returns the product. Once the product is returned, there is a mandatory testing process to determine if the product was faulty. If it is found to be faulty, it is repaired and the refurbished product created is transmitted to another channel for distribution. If no fault is found, Cisco currently ships the product back to another existing customer. They follow this policy till the product is returned 3 times (the three strikes policy) at which time they salvage or scrap the product. The aim of this project is to determine if this is the optimal policy in terms of profit maximization and evaluate the key drivers of this policy and look at the sensitivity of each of these parameters to the policy, with the objective of making an informed assessment on this policy.

**Parameters**

The following are the parameters which have been identified based on an initial discussion of the problem.

* Probability of item being shipped back.
* Probability of finding an error given an item is returned back to Cisco.
* The Gross Margins on the product.
* Shipment cost.
* Mandatory testing cost.
* Repair cost.
* Salvage cost.
* Refurbished value.

**Methodology**

The following problem can be modeled using a MDP. Since the probabilities of shipping could vary by stages ( chance of being returned first time versus returned the third time) this non stationary process can be modeled using a finite discrete MDP. We can list the parameters required to model this as follows.

**States:** The states can be categorized as the following:

1. - Original product.
2. - Product Accepted by customer.
3. - Product returned by the customer which has undergone mandatory testing. At this state we make a decision whether to repair/ reship the product based on finding a defect or to scrap the product.
4. - Product that has been repaired and refurbished.
5. - Product has been scrapped

**Model:** These states (1-4) are replicated for every stage in the problem to build out the underlying Markov chain governing the process. States 1, 3 and 4 are absorbing and the only transient state being 2, where we make a decision to either SHIP or SCRAP the product. The below figure explains the dynamics of the process.(Blue lines are the consequence of shipping and the red lines show the system process if we decide to undertake the action of scrapping.)

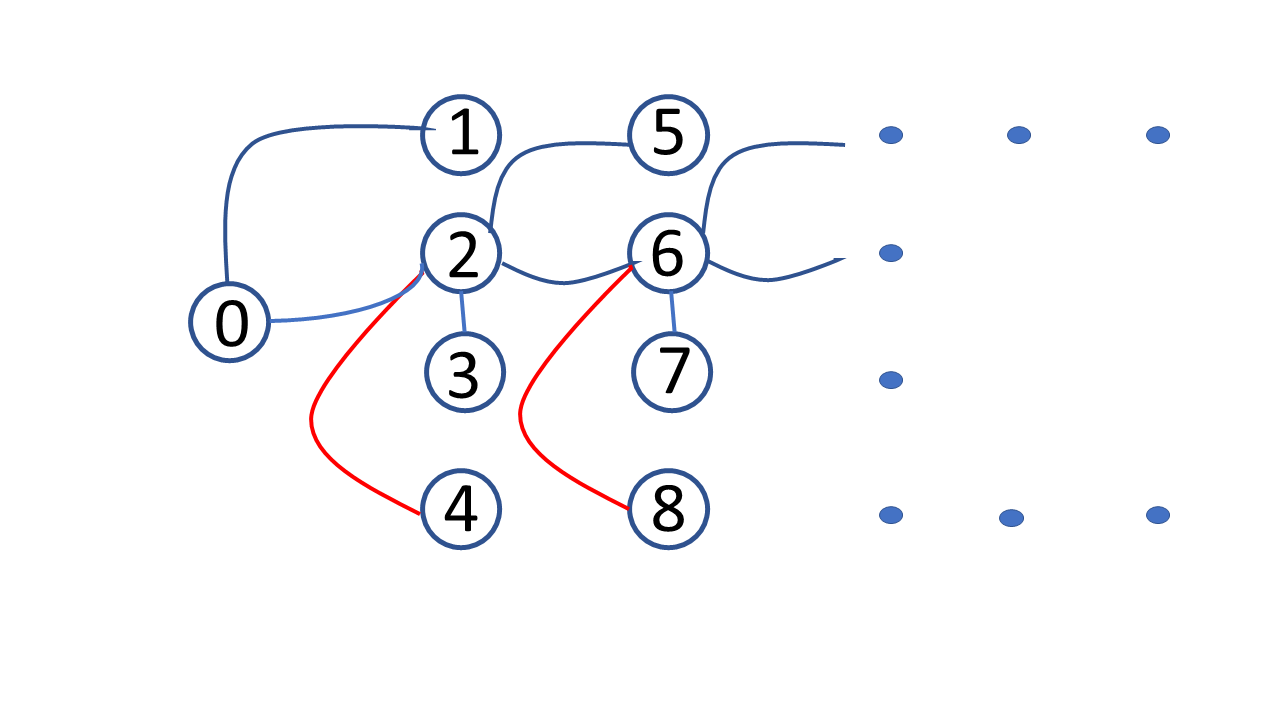


Figure 1: MDP Schematic for shipment process

**Deliverable**

* Create a MDP model in Python to model the shipment process and perform a sensitivity analysis of each parameter.

**Macro Portfolio Optimization Problem**

Cisco as an entity can be sub divided into 13 business entities (BEs). Cisco corporation has to decide how much resources to allocate to each BE in order to maximize the profit of the whole corporation. The idea is to determine the allocation of resources to each BE which maximizes the corporation's bottom line. Each BE has a return attribution based on the its individual contribution to the corporation's bottom line as extracted from the balance sheet/ booking data (Global fulfillment and Planning).

**Methodology** This problem can be abstracted into a portfolio optimization problem. A suitable proxy for returns can be a historical time series on qoq/yoy change in the BE's revenue contribution (from the balance sheet / fulfillment data) which can be used to compute an expected return for the BE and a variance around it. We can hence model the different BEs as different asset classes with a given expected return/risk characteristics. The objective thus becomes maximizing return for a given level of risk ( or minimizing risk for a given level of return) by choosing the optimal weights to invest n the portfolio of BEs. These optimal weights signify the attribution of resources to each of these BEs.

**Deliverable**

The initial assignment would entail building a simple Markowitz mean-variance efficient frontier with a Capital Allocation Line (CAL) to familiarize myself with the use of the optimization and data visualization modules in Python. Eventually a module implementing copulas to account for multivariate correlations will be developed in python to build a correlation structure which is an input to this problem.

Also a CVaR and MAD model to calculate risk will be delivered.

**Inventory control Problem**

The problem described in the previous section is resource allocation problem from a macro perspective. The analogous problem originating at the other end of the hierarchy is a bottoms up resource/component allocation problem where one decides to allocate components to various products based in the individual demands of the products subject to constraints such as order fulfillment, lead times, tardiness etc, with the objective of maximizing a parameter such as revenue, net income or minimizing costs (excess inventory etc).

**Methodology**

Currently, this problem is being handled for a deterministic setting where a given demand is assumed (or acquired) from a forecast and is used as a deterministic parameter input for the above problem.

In reality demand is stochastic and this methodology needs to be extended to include the stochastic nature of demand, supplier lead times and shipping. The problem can be formulated as a multistage stochastic program, where one begins by making an initial planning decision and subsequently at the end of each stage/period one can subsequent decisions and/or take actions based on the uncertainty which has already elapsed up until that epoch. The overall objective of the problem can thus be expressed as the sum of the nested expectation of the objectives of each stage, where each stage is linked by a set of constraints.

Alternately, this problem can also be posed as a MDP where actions taken at every stage can lead to different states at a given cost. The underlying uncertainty can be modeled as a Markov chain/process and the revenue/ cost piece can be overlaid as a reward process thus forming a MDP, which can be solved using backward recursion.

**Uncertainty modeling** Since the actual demand could be under or over the planned demand, the following are the list of levers which can be used to model the actions emanating out of the uncertain outcomes, that can be incorporated in the multistage model for cost minimization.

* Incur penalty and renegotiate contract with customer (by dropping price) in case of overage.
* Incur penalty and delay shipment to customer in case of underage.
* Incur penalty and renegotiate contract with suppliers in case of overage (ENO inventory build up).
* Incur penalty (pay more or maybe increase order size, pay extra shipping) to get component shipments from suppliers in case of underage.
* Incur freight costs from ground- air to deliver products faster ( to maintain service levels)

The uncertainty pertaining to demand, shipping and supply can be modeled using apriori distribution simulation, forecast scenarios or any other viable statistical technique which could be used to create scenarios for the optimization problem.

**Deliverable**

The above problem will be formulated as a multistage mixed integer stochastic optimization problem. It can then be abstracted to a two stage problem with an initial decision and recourse action and a framework/methodology to solve will be suggested for future implementation.

**Transformers Problem**

There are ‘N’ transformers in a system which fail randomly at some given failure rate. The system functions with ‘k’ transformers in the system (0<=k<=N) but the costs to operate the system are non linear and increase as the number of transformers in the system fall below the target number of transformers (N\_t) required to generate the required output. (One can think of these as congestion costs and the price one has to pay to buy power from the spot market to satisfy demand) One can order replacement transformers which have lead times depending on costs to acquire them. Given this scenario this problem creates a MDP model that minimizes the operating and capital costs of the whole system over time, by solving for the optimal policy for the acquisition of transformers dynamically over time.

In the supply chain world, such a model can be framed as a special case of an inventory problem. In the language of inventory problems the safety stock of replacement transformers is the “inventory” and failures are “demands.” The transformer acquisition model is an inventory problem with the following four characteristics.

* Any unserved demand is fully backlogged.
* There is a multi-period lead time.
* The inventory serves multiple demand classes, characterized by diﬀerent shortage costs.
* The demand process is fairly complex: demands are state dependent and correlated.

**Methodology**

From literature review, this problem can be formulated and solved as a MDP. Given the dimensionality of the problem, there are variety of ways to solve it using approximate dynamic programming:

* Value function approximation
* State space reduction
* Real time Dynamic Programming

The method chosen to solve this was State space reduction. The state of the system can be characterized by the tuple of (transformer id, age of transformer). The system state would then be given by n- tuples defined above. To abstract and minimize state space, the system state was redefined by the tuple ( mean age, no of transformers in the system). This reduction in state space can effectively help us solve this problem exactly as an infinite horizon MDP, once we can determine a Transition Probability matrix and a Reward matrix. Computation of these two entities entails transformation from the redefined state space to the original state space and back to be able to get a one to one congruence between the transitions and rewards from the original state space to the reduced state space.

**Formulation:**

In a T stage sequential decision making problem, at each decision epoch, the system occupies a ‘state’. Let S denote the possible system states. If at some decision epoch t the decision maker observes the system in state st ∈ S, he/she may choose an action ast among the set of feasible actions in st, represented by set Ast. As a result of choosing ast ∈ Ast in state st at time t, the decision maker receives a reward Rt(st, ast) : S × Ast → R and the system state in the next decision epoch is determined by the probability distribution pt(·|st, ast). The rewards are not influenced by future actions and their value or expected value is known prior to choosing an action. The expected value of rewards in state st and action ast is given by:

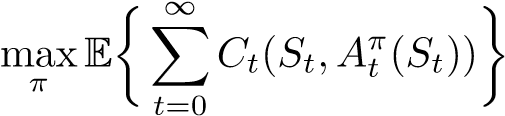
*Rt*(*st,ast*) = ∑*Rt*(*st,ast,jt*+1) · *pt*(*jt*+1|*st,ast*)*, j*∈S

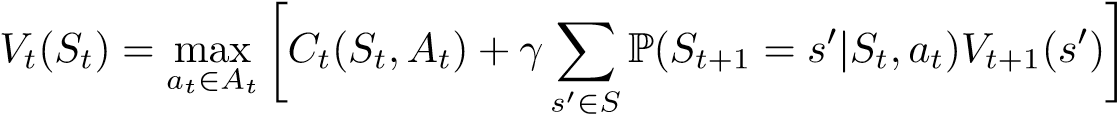
where, *Rt*(*st,ast,jt*+1) is the value of the reward received when action *ast* is taken with the system being in state *st* at time *t* which takes the system to state *jt*+1 at time *t*+1. The transition probability function is given by *pt*(*jt*+1|*st,ast*) : S × *Ast* × S → [0*,*1].

A decision rule prescribes a procedure for selecting an action at any given decision epoch. It can depend on the current state or the history incorporating all the past information. Let a decision rule at epoch *t* be denoted *dt* : S → *Ast*. This means for each *st* ∈ S*,dt*(*st*) ∈ *Ast*. Generalizing this, if *dt* is a function of the history *ht* = {*s*0*,as*0*,s*1*,as*1 *...,st*−1*,ast*−1*,st*}, then *dt*(*ht*) ∈ *Ast*. We assume that the actions are deterministic when the state or history is provided. A deterministic history dependent rule *dt* maps H*t* into *Ast* subject to the restriction that *dt*(*ht*) ∈ *Ast*. If H*t* denote the set of all histories *ht* then:

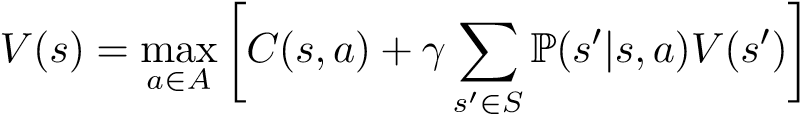
H*t* = H*t*−1 × *Ast*−1 × S

The rewards and transition function now become a function of S and H*t* after *dt* is specified. A policy or strategy specifies a decision rule to be used at any decision epoch. It is denoted *π* which is a sequence of decision rules {*d*0*,d*1*,...,dT*−1}. If the system is in state *s*0 at *t* =0, then the above sequential decision problem can be formulated as the maximum value of the sum of rewards across all policies Π, given by:



where:

This problem is a steady state stationary problem. Therefore, by letting *V* (*s*) = lim*t*→∞ *Vt*(*St*) assuming we have a limit, we obtain:



The function *V* (*s*) is equivalent to solving the infinite horizon problem. Also if the decision rule is stationary (i.e (*π*0 = *π*1 = *πt* = *π*),

*V π* = (*I* − *γ*P*π*)−1*Cπ*

The above equation can be solved to get the value functions at each state. The Transformers problem has thus been formulated as an infinite horizon problem to be solved exactly given a transition probability and a reward structure as prescribed in the recursive equations above.

**Deliverable:**

The above problem was formulated as a infinite horizon MDP with state abstraction as described in the methodology. The action set is kept fairly limited as the ability to buy transformers in any given time period is limited based on the prohibitive costs.

There are two main assumptions in this problem as applied to the formulation and implementation:

* A constant failure rate ( exponential failures) is assumed throughout this analysis. This corresponds to the flat portion of the bath tub curve when t comes to the failures of the products ( and this does not deal with the early and late stage life of the product)
* Also the cost of the transformer (value of the product) remains constant throughout the analysis period. This means that unlike products which might be more expensive when introduced and become cheaper as time goes, the value of the product (transformers here) remains constant throughout its life cycle.

The code is divided into two modules:

Module 1 calculates the transition matrix which determines the states one can go to given an initial state (a,N) based on simulation of the abstracted states to its original state and then figuring out the random possibilities of losing transformers. It also computes the associated number of transformers in the system with every failure which is useful to compute the rewards(costs) as they depend on the number of functional transformers in the system.

Example of the state space abstraction: Let us say the system has 4 transformers of the ages -6, 0, 6 and 24 units respectively. The original state space is represented by the tuple:

{(1,-6), (2,0), (3,6),(4,24)}, given by transformer IDs and age respectively. This is abstracted to {4, 6} in the model where 4 is the number of transformers and 6 is the average age of the system.

The action space is the decision to buy ‘x’ number of transformers at a lower cost ( corresponding to a higher lead time) and ‘y’ number of transformers at a higher cost (corresponding to a lower lead time). The output of the program prescribes a policy which is a specific action corresponding to any given state in the system.

The transition probability matrix is computed by converting the abstracted state to an original possible state by simulating a possible state given number of transformers and an average age of the system. Then we randomly create possible failures of 1 to N-1 failures in the system and at each pass calculate the mean age of the system This gives us the possible number of transitions to various state from any given initial state.

For example in the above example, if the initial state is (4,6) the possible combinations could be:

(-6,0,6,24) as in the previous example, (6,6,6,6) is another possibility, ( 8,8,8,0), (4,4,4,12) (-6,-6,-6,42) are some other possibilities. For each of these we then randomly remove 1-3 transformers and recalculate the average age of the system. In the above example if the transformer of age 24 fails, then the mean age of the system at the next time period would be (-6+0+6)/3 +1 so the new state would become (3,1)- three transformers and mean age of 1 years (assuming we don’t buy anything in that time period). We can therefore compute p(s\_{t+1}| s\_t, a\_t} as we know the possible actions In the above case if the action was t buy 1 cheap and 1 expensive transformer, the new state would be

(-6+0+6-6-3)/3+1 ~(5,-2) as now 2 more transformers are in the system, as the lead time for a cheap and expensive transformers are assumed to be 6 and 3 time units respectively.

We also compute the actual number of transformers in the system that can fail or in other words the number of active transformers in the system. Any transformers that are non functional i.e. whose age is <0 cannot fail. The costs are calculated based in the initial state of the system i.e. before any failure occurs. The failures all occur at the end of any stage which means the transformers have been assumed to have been fully utilized in the period. (in practice there might be a failure at any given point and on an average one can say that a functional transformer has been functional for half of the period. Given the cost function, one can modify it to include fractional number of transformers if one wishes to. That has not been done here) Therefore, before any failure we calculate the number of functional transformers in the system as the operating costs depend on the number of functional transformers in the system. In the same example of (-6,0,6,24), the number of functional transformers in the system would be only 3 ( as out of the -6,0,6 and 24, the -6 is non functional) even though there are four transformers as defined in the state. The cost matrix keeps track of the number of active transformers in each run.

Module 2 takes this transition matrix and the associated cost matrix, builds the transition probability and the reward matrix which is fed into the infinite MDP solver from the mdp toolbox to determine the optimal policy. Given we have the probabilities of state transitions and the average number of transformers in the system based on each simulation run, the TPM is calculated now by multiplying the probabilities of 1-N-1 failures ( which given an constant failure rate = exp(-lambda)) times the probabilities of being in any given state conditional to 1-N-1 failures. Similarly the reward matrix is populated by taking the expectation of the costs based on the number of functional transformers in each simulation run.

The output of the program prescribes a policy which is a specific action corresponding to any given state in the system.