Optimization of Stochastic Inventory Control with Correlated Demands

Roger Lederman

Honors Thesis in Computer Science

Advisors:
Amy Greenwald
Aaron Cohen

1 Introduction and Motivation

An application for Intelligent Agents that has recently received attention is Supply Chain Management. The use of Intelligent Agents makes possible new approaches to SCM problems, ranging from Sales Decisions, and Assembly and Distribution, to Inventory Management. This paper will explore the inventory management problem in the context of an intelligent SCM agent.

We focus on a difficult inventory problem, that of producing optimal ordering policies for components, when the demand for components overlaps products. The combinatorial nature of the cost-minimization problem makes it difficult to optimize analytically. The goal of this study is to evaluate the success of search techniques in finding optimal inventory policies.

The background is drawn mostly from Operations Research, where work has been done to characterize the optimal solution to the inventory problem with correlated demands. We will present this background, highlight the difficulties in using existing methods to manage inventory when decisions must be made quickly, and evaluate the methods that an agent might use to overcome these difficulties. Our results are obtained through simulation of an Assemble-To-Order (ATO) System.

As a framework, we use the problem of TAC (Trading Agent Competition) 2003, where an SCM problem for agents has been defined. TAC is a competition where agents compete as PC manufacturers who must sell products, consisting of configurations of common components, based on market predictions. Inventory Management is an essential agent activity.

2 Model

In this section, we will describe the way in which we have attempted to model the inventory problem for a Supply Chain Management Agent. We will introduce the inventory mechanism and its notation, and present some of the classical results of inventory theory from which our strategies are built. Sections three and four will highlight some of the specific problems that arise in the more complicated environment in which our agent will operate. We will then describe prior work that has been aimed at solving these problems [8,9,12] before presenting our own strategies and results.

2.1 Assemble-To-Order

It is possible to model the TAC agent inventory problem as a cost-minimization problem in an Assemble-To-Order system. An ATO system is one in which demand is received for products, and supply is ordered in terms of components. Stock is held only for components, and products are assembled when an order is filled. Assembly is instantaneous. [12] This resembles the TAC inventory problem, if we define filling demand as delivering the necessary components, not to the customer, but to the assembly system. In this case, we need only deliver the right group of components, so no assembly time is necessary. We must deliver the components in a complete group, so the system looks the same as an ATO. In TAC, there are multiple sets of goods that could fulfill the same product demand, but we will ignore that substitutability for now.

Definition: ATO

- Components, $i \epsilon \{1, \ldots, m\}$

- Products $j \epsilon \{1, \ldots, n\}$, st. $j \subseteq \{1, \ldots, m\}$
- Demand Distribution: $\varphi(\xi = d)$ at time t
- Inventory: $y_i(t) =$ Stock of i on hand at time t
- Penalties: p^j, h_i cost of product shortage, excess inventory

2.2 Inventory Management

Each day, demand is generated by each of our demand processes, and we are thus faced with a quantity of each product that we are required to fill. In an Assemble-To-Order system, stock is stored in components. When an order for a product comes in we have to take each of the required components out of stock and assemble and deliver the product. An inventory policy aims to have all of the necessary components in stock, so that orders can be filled from inventory that is already held on hand when the orders arrive.

However, the overriding goal is to minimize costs, and the holding cost is thus a prohibitive factor that will limit the size of a stock that is profitable to hold. With limits on the quantity of inventory that is held, it is possible that we will not be able to meet all of a particular day's demand from the inventory that we have on hand. In this case, a product order will need to be backordered while we wait for the necessary components to arrive in stock.

Typically when orders are backordered, it will be among our highest priorities to fill these orders. The mechanism that we have implemented for filling orders in this study is First-In, First-Out (FIFO), which assigns priority to the earliest arriving orders. If backorders exist, then they will have to be filled before the day's demand (actually, we do not have a strict FIFO, as we make anything that we are capable of making at the time). In reality, there will be an Assembly component to the agent that will decide which orders are filled first, and we will provide components as they are needed. We will assume a more pressing need for those components that the Assembly Component asks for first, so FIFO seems like a good model of the Agent's inventory service mechanism.

When the day's demand is received, it is queued for fulfillment behind the backorders that have been carried over from the day before. Thus, when backorders exist, our physical measure of inventory on hand will not accurately represent our ability to meet incoming demand. We introduce a new term, Net Inventory, n_i, to represent our stock level of physical inventory for a component, less the quantity of that component required to meet product orders that have already been backordered.

In defining backorders as demand that we are incapable of filling on its arrival, we are making the assumption that it is desirable to fill orders as soon as they arrive. However, actual orders will be for some specified future due date. Of course, it is possible to reduce the actual system into our system by waiting until the due date to fill orders, so there is not necessarily any problem with our mechanism. However, this approach ignores the value of advanced information to the inventory policy. Still, we must consider that our demand is actually demand imposed on us by the Assembly component, so there is a time of assembly as well. This means that the actual lag between customer orders and the time at which the component is needed will be *due date lag - assembly time*.

We choose, instead, to model the inventory system as an ATO where demand comes from the Assembly component, and is to be filled immediately. To take advantage of advanced information, we will consider some portion of the demand deterministic, based on knowledge of the upcoming assembly schedule, and some portion to be stochastic. The stochastic portion will be smaller than in a model that does not account for advanced knowledge and can thus be predicted more accurately, lowering costs. For

the purpose of this paper, we will look only at how to optimize the inventory levels necessary to prepare for the stochastic portion of demand. Since we are concerned with the risk resulting from uncertainty of demand when determining the optimal stock level, the components needed for filling deterministic demand can be added to our solutions without affecting their optimality.

We will, however, pay close attention to the delay experienced in obtaining components from the supplier. The notion of leadtime will be central to our solutions. When a component is ordered from a supplier, there will be some delay, possibly stochastic, until it is received. As a result, we must plan ahead with this delay in mind, realizing that we must order today the inventory not only for today's or tomorrow's demand but possibly for demand due well in the future. In making decisions based on this type of horizon, it becomes important to know not only how much inventory is physically on hand, but how much is scheduled to arrive in the near future. Thus we add to our notation the term x_i, representing the quantity of component i that is currently due to us from our suppliers.

A common assumption when modeling inventory is that leadtimes do not cross, meaning that all components ordered at time t will arrive before those ordered at time $t+1$. Again, this is not an assumption that will necessarily be accurate, but a heuristic that helps in establishing optimality. A consequence of this assumption is that we can count all orders that have been placed with the supplier as *orders due* when making our current inventory decision, because they are guaranteed to have arrived before the components that we are currently ordering.

Given these modeling assumptions, we can use the following variables to represent the state of our ATO at a time t:

- $y_i(t) =$ Physical Inventory at time t
- $w_i(t)$ Quantity of Component Committed to Backordered Products
- $n_i(t)$ Net Inventory at time t
  - $n_i(t) = y_i(t) - w_i(t)$
- $x_i(t)$ Outstanding orders for component expected from supplier at time$t$

We will use these indicators to determine, based on our policy, the optimal amount of each component to order at time t.

2.3 Maintaining A Basestock

The policy that we will implement for managing our inventory is called an "Order-Up-To Policy", and is dependent on establishing on optimal stock level called a "basestock level", which the policy will then try to maintain. The optimal basestock level, s^*, is generally not the quantity of physical inventory that we have on hand. Rather, s^* will be a target level at which we will attempt to keep our sum of Net Inventory and Outstanding Orders.

The policy operates by (if possible) filling demand from inventory on hand, and then placing a replenishment order for the same quantities of components that were used up in filling the day's demand. The actual quantity ordered is thus dependent on both the inventory position (on hand, backorder, and due from suppliers) at the start of the day and the day's demand.

If we first fill the day's demand (or backorder what we cannot fill), the ordering decision for day t becomes:

Order Quantity for day t, $z_i(t) = s^* - n_i(t) - x_i(t)$

2.3.1 Optimality of Order-Up-To Policies

The basestock strategy is meant to ensure that there are always enough components in the system (in inventory or on the way) to fill the demand that is expected between today and the time at which any parts ordered today will arrive. The following analysis will consider only a single time period t, with the purpose of proving that, given a distribution for demand, there exists an optimal level at which to keep your holdings so as to minimize expected costs. Furthermore, we see that this optimal solution is dependent on the respective costs of ordering too little or too much of a component, and attempts to balance the expected costs of each.

Focusing on a system of only one item (one component and one product), if we follow an order-up-to policy, then the cost for each period can be described by the function:
$g(d, y) = h \times (y - d); y > d$
$\quad\quad\quad\quad p \times (d - y); y \leq d$
The expected costs for a given stock level can thus be expressed by C(y):
$C(y) = \int_0^\infty g(\xi, y)\varphi(\xi)\, d\xi.$
Splitting holding and shortage costs, we get:
$C(y) = h \int_0^y (y - \xi)\varphi(\xi)\, d\xi. + p \int_y^\infty (\xi - y)\varphi(\xi)\, d\xi.$
Later on, we will look at evaluating this to obtain an expectation of our costs, but to show that the order-up-to policy is optimal (and to compute the optimal basestock, for that matter), we only need to find a minimum to C(y), so we set the derivative to 0.
$\frac{dC(y)}{dy} = h \int_0^y \varphi(\xi)\, d\xi. - p \int_y^\infty \varphi(\xi)\, d\xi. = 0$
Integrating over the pdf, ξ, we get the cdf, Φ, and because $\int_0^\infty \varphi(\xi)\, d\xi. = 1$,
$h\Phi(y^*) = p[1 - \Phi(y^*)] = 0$, and thus $\Phi(y^*) = \frac{p}{p+h}$ [3]

We obtain the somewhat intuitive answer that the optimal basestock in a stochastic setting is one which sets the probability of overordering to the ratio of the *shortage* penalty to *holding + shortage* penalties.

This proof will hold for a general demand distribution, though we will assume a Poisson demand process (i.e independence of demand between time periods). The requirement for optimality to hold is the convexity of the expected cost curve. This ensures that there will be a global minimum and is true provided non-decreasing holding and shortage costs. Thus there exists an optimal order-up-to policy for a very general case of the single-item problem.

3 Single-Item Solutions

The case considered in the above proof, where only a single period comes into consideration is known as the "Newsvendor Problem" (so-called because demand for a particular newspaper lasts only for a single day), and has been well studied. As in the Newsvendor case, we will begin our study of Inventory Control policies through time, by examining the single item case. The calculations are a good deal simpler than

in the TAC problem, but it is a revealing special case of an ATO.

If $C(s, \varphi(\xi))$, is the cost function, given a basestock y, and a (exogenous) demand function, we can formulate the single-item optimization problem [12] as follows:
$C(s, \varphi(\xi))$ = minimize $E[\sum_{t=0}^{T}[hy_t + pw_t]]$

subject to

$z_t + y_t = s$

$z_t + w_t = d(t)$

$w_t, y_t, z_t \geq 0, \forall w, x, z$

We assume that the demand and cost functions are stationary, so the optimal y remains the same throughout all periods. For the single-item case, this optimization problem can be easily solved as a linear program. However, we will detail algorithms for determining the optimal basestock level y that will be useful in our approximations of the larger multi-product optimization problem.

3.1 Zero Leadtime

The first case is the case of zero leadtime. This is a special case in itself, and has an optimal solution for any configuration of components to products. When leadtime is zero, we can satisfy demand immediately as it is realized. Assuming unlimited capacity, there is no need to follow a policy in this case. This approach will remain optimal with any type of stochastic demand or correlation between components, so we will have no need to look any further at the zero leadtime case.

3.2 Deterministic Demand/ Deterministic Leadtime

The deterministic demand/deterministic leadtime case collapses to a zero leadtime case, where decisions need to be made a leadtime cycle before the due date. Although we cannot instantly fill demand, as we do in the zero leadtime case, we do have full knowledge of the demand schedule, and can thus order ahead of time and never have to face uncertainty.

This solution is useful in conceptualizing the way in which a basestock is affected by leadtime. Although, with deterministic demand we should be able to maintain a physical inventory of 0 (by ordering exactly the amount necessary to fill demand), our optimal basestock will be equal to the *leadtime demand*, and will consist entirely of outstanding supplier orders.

Leadtime demand is an important concept in the functioning of a basestock policy. We have established that the goal of a basestock policy is to maintain a level of components in the system sufficient for meeting the demand that we expect during a leadtime cycle. The strategy we have described for the deterministic demand/deterministic leadtime case equates to maintaining a basestock of $\lambda * l =$ *leadtime demand*.

Algorithm ComputeDeterministicBasestock
Inputs:
Demand Rate λ
Leadtime l
Outputs:
Optimal Basestock Level s^*

1. LeadtimeDemand = $\lambda * l$
2. Return s^* =LeadtimeDemand

3.3 Stochastic Demand/Deterministic Leadtime

With stochastic demand, the goals are the same as in the deterministic case. If we will be fulfilling an order on day t, then we must order that quantity on day $t - l$, since it will take l days to arrive. If we do this properly, we can fill the order from the inventory on hand (most of which will have arrived at the start of the day). If we keep $X_i + N_i$ equal to the leadtime demand, then our inventory on hand can be used to fill all orders prior to the arrival l days from now. The problem in the stochastic case, is that we do not know exactly what the leadtime demand will be. We could carry the expected leadtime demand, but if we face very high backorder costs or very high holding costs then it will be to our advantage to order more or less so that we increase or decrease our probability of overordering or underordering to avoid the larger cost. The basestock level that optimally balances these costs can be obtained using the inverse cumulative distribution function of the leadtime demand distribution.

The goal is to set:
$P(\text{Overordering}) * \text{CostOfOverordering} = P(\text{Underordering}) * \text{CostOfUnderordering}$
If ϕ is the cumulative distribution function of Leadtime-Demand, then, in the single-item case, this translates to:
$s^* \leftarrow \phi^{-1} \frac{b}{b+h)}$
In the single item case, b is the unit costs of underordering and filling demand late. h is the unit cost of overordering and holding excess inventory.

Algorithm ComputeStochasticDemandBasestock
Inputs:
Poisson Process with rate λ
Leadtime l
Late Penalty p
Holding Cost h
Outputs:
Optimal Basestock Level s^*

1. LeadtimeDemand $\leftarrow$ PoissonProcess with rate $\lambda * l$
2. UnitCostOfUnderordering $\leftarrow p$
3. UnitCostofOverordering $\leftarrow h$
4. DesiredRiskOfExcess $\leftarrow$ UnitCostofUnderordering/(UnitCostOfUnderordering + UnitCostOfOverordering)
5. Let $\phi =$ InverseCumulativeFunction of LeadtimeDemand
6. $s^* \leftarrow \phi(\text{DesiredRiskOfExcess})$
7. Return s^*

3.3.1 A basestock example

Problem: We have one item which we can buy from a supplier for 10 dollars at a deterministic leadtime of 3 days, and can sell for 20 dollars. Daily demand comes in the form of a Poisson process with a rate of 20 items per day. If more items arrive from the supplier than there is demand for, we can hold items in inventory at a cost of $\$5$ per unit per night. If our net inventory (Inventory on hand - Customer Backorders), is less than the days demand, we are charged a shortage price of 8 dollars for every unfilled order (backlogged) at the end of the day. What is the optimal basestock level to maintain?

Solution: With deterministic leadtime of three days, the leadtime demand, ϕ will come in a Poisson process with rate of 60.
DesiredRiskOfExcess = $\frac{8}{8+5}$ = .6154
BaseStock = $\phi^{-1}(.6154)$ = 62 Items.

Figure 1: Ending Balance and C(s)
Refer to Figure 1 in the appendix. The graph shows results from simulations of basestock policies from 40 to 80, under the above conditions, with the optimal at 62. Notice the holding costs increasing with s, and the shortage penalties decreasing.

3.3.2 Steady State Analysis

Steady-State analysis illustrates the relationships between all of our state values, and translates into an alternate representation of the objective function and the optimal basestock level. The ATO system can be modeled as an M/G/∞ queueing system. This means that inputs are Poisson processes , service times are generally distributed, and there are an infinite number of servers. This is a fitting description of our system, where demand from customers constitutes the entry of an order into the system, and arrival of components into stock constitutes a departure. Thus, all orders due from the supplier can be considered jobs in our system.

Since we place a replenishment order with our suppliers whenever we fill demand, there is a Poisson arrival of jobs into the system. Furthermore, an infinite server queuing system has the characteristic that every job arriving is immediately serviced, and thus there are no queues, only jobs being serviced. The effect of this is that the time needed to service a job (receive an order from the supplier) is independent of the number of jobs being serviced. Since every order is given a leadtime, and that length of time represents its service time, regardless of the number of other orders due, an infinite server system is the appropriate model. (Future work will be to model correlation in our leadtime distribution, so that the service times are no longer generated by independent distributions, though this does not necessarily mean reducing the number of servers).

The M/G/∞ queueing model provides a complete description of the variable x_i. The variable X_i can be used to represent the steady-state level of jobs in the system and thus the expected value of x_i. This value should equal $\frac{\lambda}{\mu}$, where λ is the arrival rate, and μ is the service rate. λ is simply the rate of the Poisson process generating demands and μ is equal to $\frac{1}{E[l]}$, since l, the *leadtime*, is the time for any service. Thus, the number of orders we expect to have due from the supplier at any time is: $X_i = \lambda_i * E[l_i]$

The above equation implies that the expected number of outstanding supplier orders is not affected by the level of s^* that we set, provided we operate an order-up-to policy. What our policy will decide, are the respective levels of on-hand inventory and backorders. We can express these in terms of X_i and s_i.

This allows for the steady-state formulation of other system indicators as described by [8].

- N_i is the steady-state distribution of Net Inventory for item i.
- I_i is the steady-state distribution of Inventory on hand for item i.
- B will represent steady-state distributions of backorders, with B^j representing product backorders for product j, and B_i representing the number of product backorders caused by a shortage of component i. In the single item case, we will use B_i to represent both the product and the component.

Since these distributions are dependent on our policy s, we will use the notation $N_i(s), I_i(s), and B_i(s)$ to represent that steady-state distributions resulting from a specific policy.
By definition of the ordering decision under an Order-Up-To policy, $N_i(s) = s_i - x_i$. Costs are bases on I_i and B_i, which together make up the Net Inventory. When Net Inventory is positive, we will be charged holding costs on excess inventory. When Net Inventory is negative, we will incur shortage penalties. Thus, we have the identities:

- $I_i(s_i) = [s_i - X_i]^+$
- $B_i(s_i) = [X_i - s_i]^+$

Since X_i is determined exogenously, we can clearly see the way in which our policy s determines costs. The actual costs can be computed by integrating over the distribution, X_i, to obtain:

- $I_i(s_i) = \int_0^{s_i} (s_i - x_i)\, dx_i$
- $B_i(s_i) = \int_{s_i}^{\infty} (x_i - s_i)\, dx_i$

The expected costs in the steady-state
$C_i(s_i) = h \int_0^{s_i} (s_i - x_i) P[X_i = x_i]\, d_{x_i} + p \int_{s_i}^{\infty} (x_i - s_i) P[X_i = x_i]\, dx_i$
closely resemble those in the Newsvendor problem,with the distinction that we compare demand over a leadtime to supply + supply due, rather than a day's supply and demand. We can optimize s_i using the inverse cdf of X_i the same way we minimized costs by computing y from the inverse cdf of demand in the Newsvendor problem.

3.4 Stochastic Leadtime

When both demand and leadtime are stochastic, the optimal policy will be dependent on the leadtime demand. When demand is Poisson distributed, the expected arrivals in consecutive days are independent random variables. Because Poisson processes can be split and aggregated into Poisson processes, the distribution for the demand over a leadtime consisting of some uncertain number of periods is exactly a Poisson distribution, with a rate of $\lambda * E[l]$. Thus, the algorithm above can be easily adapted to the case of stochastic leadtimes. Although this is a trivial change from the case of deterministic leadtime, we present the algorithm because it will be called by algorithms developed later:

Algorithm ComputeBasestock
Inputs:

Poisson Process with rate λ
Leadtime Distribution $\varphi(L = l)$
Late Penalty p
Holding Cost h
Outputs:
Optimal Basestock Level s^*

1. LeadtimeDemand $\leftarrow$ PoissonProcess with rate $\lambda * E[l]$
2. UnitCostOfUnderordering $\leftarrow p$
3. UnitCostofOverordering $\leftarrow h$
4. DesiredRiskOfExcess $\leftarrow$ UnitCostofUnderordering/(UnitCostOfUnderordering + UnitCostOfOverordering)
5. Let $\phi =$ InverseCumulativeFunction of LeadtimeDemand
6. $s^* \leftarrow \phi$(DesiredRiskOfExcess)
7. Return s^*

4 Assembly System

Another special case of an Assemble-To-Order system is an assembly system. In an assembly system, there is only one product being produced, but it must be assembled from several components. The solutions are considerably more complex than in the newsvendor problem, but are more intuitive than in the Multi-Product case.

4.1 Deterministic

The deterministic case for an assembly system is not much more complicated than in the single-item case. Complexity in assembly systems arises from reliance on the joint demand of components to determine expected penalty costs. As in the single-product system, in the deterministic case of an Assembly system, the optimal ordering policy will create zero penalty costs, so we do not need to worry about joint demand. For each component in the Assembly System, the (Order-Based) policy is to determine the leadtime demand for the product and multiply that quantity by the quantity of the component that is needed for each unit of product demand that is filled. This allows us to solve a Newsvendor-like problem for each component.

Components in an assembly system are complementary. When supply of one is increased, supply of the others should be increased as well, to maintain optimality. This has no effect on the deterministic optimal policy, but will be important in the stochastic case. If a product consists of two components, we always want supply to be equal. If supply of one component exceeds that of the other, that supply must be held as excess, since we cannot yet deliver the product. Thus, even though the joint policy determines the balance, we only need to look at one decision variable to determine the supply of both components. This allows a single-item approach to perform optimally.

4.2 Deterministic, Different Leadtimes

The leadtime demand can be calculated separately for each component in an assembly system if they have different deterministic leadtimes. Again, since penalty costs are avoided altogether, it is not necessary to consider joint demand. This allows us to solve a Newsvendor-type problem for each component.

4.3 Stochastic, Deterministic Leadtimes

The stochastic problem with deterministic leadtimes can be solved by computing the single-item solution for the product. Essentially, we treat the entire group of components as one item. We can ensure that there will always be the same stock level for each component present, so we can think of the product as one item and order the components together. Thus, the holding cost will be the sum of all holding costs and the backorder cost will be the backorder penalty for the product. We can then order that amount for each.

Algorithm StochasticAssemblySolution
Inputs:
Poisson Process with rate λ
Leadtime l
Late Penalty p
Holding Cost vector h
vector $A = a_i$
Outputs: Optimal Basestock policy, m-vector s^*

1. LeadtimeDemand = PoissonProcess with rate $\lambda * l$
2. UnitCostOfUnderordering $\leftarrow p$
3. UnitCostofOverordering $\leftarrow \sum_i h_i$
4. DesiredRiskOfProductExcess $\leftarrow$ UnitCostofUnderordering/(UnitCostOfUnderordering + UnitCostOfOverordering)
5. Let $\phi =$ InverseCumulativeFunction of LeadtimeDemand
6. For $i = 1...m$
7. $s_i^* \leftarrow \phi$(DesiredRiskOfProductExcess) $* A_i$

4.4 Stochastic, Stochastic Leadtimes

When leadtimes are stochastic, the Assembly system problem becomes considerably more difficult. With deterministic leadtimes, we had assurance that we could schedule complementary components to arrive simultaneously. This assurance allowed us to treat the entire product as a single item. However, when leadtimes are different, there is a chance that one component will arrive on time, but we will still need to hold it in inventory because another component has been delayed. In addition, when a component is delayed, we do not know whether the shortage penalty is caused entirely by that component, or if several components have been delayed. Thus, shortage costs become a function of the joint basestock policy.

We can still compute a steady-state for component shortages using only the component demand and its expected leadtime, but we are unable to predict how this will translate into product shortages without taking the leadtime distributions of the other components into account. We will give approximation methods for the costs incurred, as developed in [9], when we solve the multi-product problem.

5 Difficulties in the Multi-Product Case

5.1 Multi-Product Singleton

If there are multiple products, and each one is distinctly composed of one component, than it is possible to use ComputeBasestock to solve for the optimal policy for each item, since there is no correlation between the items or the backorder costs.

5.2 Multi-Product No Overlap

If there are multiple products, and each one is composed of a distinct set of components, then it is possible to use StochasticAssemblySolution on each product to compute a stock policy for its components.

5.3 Overlapping Component Demand

Unfortunately, even in the case of deterministic leadtime, it is not possible to solve this problem optimally using StochasticAssemblySolution if the products share components. It may seem intuitive to use StochasticAssemblySolution on each product, and then sum up the stock levels needed for the solution of each product in which the component is included. However, this approach is flawed in that it does not account for our ability to pool our risks by dealing with separate outstanding order queues for the same component as one queue. The following example of a multi-product ATO with component overlap will make this clear.

5.3.1 Risk Pooling

We have two products, A and B, each consisting of one component, C. We will assign a backorder cost of 5 to each product and a demand rate of 20 units. The holding cost for C is 3. The leadtime for C is 3 days.

The item-based solution for both A and B is calculated as follows:
Leadtime Demand = 60.
DesiredRiskOfExcess = (5/8)
$s^* = 64$
Total Item-Based stock level $= 128$

If the demand processes are both Poisson, we know that the total demand for C, from the aggregate of the processes will be a Poisson process with a rate of 40.
We can calculate a true optimal stock level for C using the aggregate demand:
LeadtimeDemand = 120.
DesiredRiskOfExcess = (5/8)

$s^* = 126$
The Product-Based policy overestimates the risk of stocking out and will accumulate excess holding costs.

5.3.2 Backorders and The Joint Cost Function

The 'Risk Pooling' example suggests that to calculate the optimal policy for each component based on independent item calculations, we should calculate for the aggregate component demand stream, rather than solving a single-item problem for each product and extending the solution to its components. However, this approach makes it difficult to assess the shortage costs for each item, as the exact shortage cost for a unit of each component must consider the penalties that could result from a shortage of each product the component is in, and in the correct proportions. In the assembly system approach, we recognized that when we underorder for one component, we incur penalties not only for failure to fill demand on time, but for the excess of other components that we now hold. The extent of these penalties is dependent on the joint stock policy. We will now look at how the optimal vector s can be calculated.

We can use the queuing approach to aid in developing an expression for the expected cost. If we have steady-state values for the $I(s)$ and $B(s)$, then we can use them to calculate expected costs [9]. For any component, the total expected costs will have a holding cost component and a penalty cost component. The holding cost component will be dependent on $I(s)$ and on h_i. The penalty cost component will be dependent on $B(s)$ and on both h_i and b^j, since component shortages result in both product shortages and excess inventory of other components.

As in the derivation of the optimal basestock policy, the cost function will consist of steady-state inventory, $\int_0^{s_i}(s_i - X_i)\,dx_i. = [s_i - X_i]^+$ times the cost of overordering, and steady-state backorders $\int_{s_i}^{\infty}(X_- s_i)\,dx_i. = [X_i - s_i]^+$ times the cost of underordering. b^j is a shortage cost applied to products, and we will thus have to assess a penalty of b^j on product shortages, which have a steady-state rate of $B^K(s)$. One component of the cost function will be this product $\sum_j b^j E[B^j(s)]$. h_i is a unit holding cost applied to components. It is clear that we will incur a cost $h_i * [s_i - X_i]^+$ which by the steady state identities is equivalent to $[s_i - X_i + B_i(s_i)]$. This cost covers the holding cost for holding inventory up to our basestock levels.

Still missing from our cost function is a term to represent the holding costs incurred on complementary components when a component is backordered. Since h_i is a per component cost, this must be component-based. The rate at which we experience shortage related holding costs for component i is $J_i(s) = \sum_{j \ni i}[B^j(s) - B_i^j(s_i)]$. $[B^j(s) - B_i^j(s_i)]$ is the difference in that rate of backorders for product j and the rate of shortages for component i resulting from demand for j. This difference represents the rate at which component i is delivered to meet demand for j, only to be held because of a shortage of another component.
[9] shows that the expected costs sum to:

$$C(s) = \sum_i h_i s_i + \sum_j \tilde{b}^j E[B^j(s)]$$

where $\tilde{b}^j = b^j + \sum_{i \epsilon j} h_i$
The backorder cost for one component is dependent on the backorder rate of other components and thus their policies, making the overall backorder cost minimization problem a non-separable function of the joint distribution.

The approach outlined in [9] is to solve an independent problem for each component to determine its optimal level. We estimate the shortage costs that a backorder of component i will cause in each product $j, i \epsilon j$ in such a way that we are assured an upper bound on each components basestock level. This

eliminates the problem of not having an accurate way of translating product demand into component demand (because of risk pooling) or component costs into product costs(due to an inseparable function for backorder costs). However, the approach is difficult to implement in that it involves evaluating $C(s)$, which is a difficult problem in itself.

We experiment with evaluating $C(s)$ through simulation. The benefits of this approach are limited by the problem that inexact cost values can disrupt the convexity of $C(s)$ and the time required to reach steady state average costs makes the algorithm undesirable for practical use.

6 Approximations

6.1 Upper Bound

In the equation for the optimal cost, the total backorder cost is calculated for products, and is a function of the product backorder rate and the product backorder cost. In order to separate the cost equation, a bound can be put on the effect that a change in one items basestock has on the total backorder costs.
Recall that $B^j(s) = \max_{i \epsilon j} B_i^j(s_i)$
An effect of this is that: $B^j(s + e_i) - B^j(s) \geq B^j(s)$. [9]
We expect the derivative of the function of Steady-State backorders due to type j demand from item i to relate to that of all demand for item by the ratio $\frac{\lambda^j}{\lambda_i}$, the ratio of demand for product j, to that of item i. (Note that if we relax the unit matrix restraint we need to multiply this by a_{ji}.)

Thus, if we take the expectation of the above inequality, we get the relationship we need between the derivatives: $\triangle_i B^j(s) \geq \triangle_i B_i^K j s_i) = \frac{\lambda^j}{\lambda_i} \triangle B_i(s_i)$
Since the derivative of the holding costs is constant with respect to s, this inequality containing the derivative of the backorder costs can be extended to the derivative of $C(s)$. From this we can obtain that $\triangle_i C(s) \geq \triangle_i C_I(s)$. [9] show that by the submodularity of C(s), this leads directly to the conclusion that $s_i^* \leq s_i^{I*}$ for all i. Thus, if we optimize s_i for $b_i = \sum_{j \ni i} \frac{\lambda^j}{\lambda_i}(b^j + \sum_{j \epsilon_j} h_j)$ we obtain a lower bound on s^*.

Essentially, we have shown that by overstating the effects of a component shortage, we can achieve a definite upper bound on s^*. The unit backorder cost is determined by two things. When components overlap products, the difficulty is to assess what portion of the backorders for a component will be expressed by a late order in each product. Furthermore, when a backorder for component i causes a penalty on an order of product j, what is the additional cost that can be expected because of holding costs paid on other components in that product that have already arrived at the supplier. These expectations are calculated by integrating over an exponential number of permutations of leadtime distributions [8], but we can only estimate when deriving closed form approximations [9]. The cost calculation associated with the above upper bound for s_i assumes that every time a component is backordered, it causes a penalty that would not have already been caused by a shortage of a complementary component.

Thus, for each component, when we compute the upper bound, we assume that:

1. The unit shortage cost for component i includes a portion of costs from each product, j, that is proportional to the expected costs of a shortage of product j caused by a backorder of component i, by the proportion of demand for component i that comes from product j.
   This means that we expect a shortage penalty to result from every backorder of product i, ($\sum_{j \ni i} \frac{\lambda^j}{\lambda_i} = 1$), and that any shortage of K that occurs when multiple components are out of stock, is counted

multiple times.

2. Each time there is a backorder of component i resulting in a shortage of product j, we expect to incur as a result of that backorder of component i, a holding cost on every item in j other than i.

The above provides an intuitive description to accompany the mathematical proof of [9]. This type of understanding is necessary in looking for a lower bound to increase the effectiveness of the closed form approximations. [9] provide no mathematical proof for their suggested lower bound, but point to extensive empirical evidence. The goal in determining a lower bound would be to understate the effects of a shortage, rather than overstate. The counterpart of our upper bound would be to assume that all of the components that are packaged in a product with component i are stocked out when component i is stocked out. This estimate is both unrealistic and too low to be effective. A tighter lower bound is suggested by [9], with justification provided in that paper. We computed this bound: $p_i = \Sigma_{j=\{i\}} \frac{\lambda^j}{\lambda^i} + \Sigma_{j \ni i, j \neq \{i\}} \frac{\lambda^j}{2 \max_{c \epsilon j}\{\lambda_c\}} (p^j + \sum_{c \epsilon j, c \neq i} h_c)$
This bound is based on the idea that if the shortages of respective components in a product occur simultaneously as much as possible, the product backorder rate will be the maximum of all components backorder rates, since no other component will cause a product shortage when this components is in stock.
The two bounds described in this section yield the following algorithms, used to initiate our search procedures:

ComputeUpperBound
inputs:
ATO System
output:
m-vector, s, of Upper Bounds on Optimal Basestock levels

1. $\forall i$
2. $\lambda_i = \Sigma_{j \ni i}\, a_{ji} * \lambda_j$
3. $p_i = \Sigma_{j \ni i} \frac{\lambda^j}{\lambda_i} (p^j + \sum_{c \epsilon j, c \neq i} h_c)$
4. $\forall i$
5. $s_i =$ ComputeStochasticBasestock(λ_i, l_i, p_i, h_i)

ComputeLowerBound
inputs:
ATO System
output:
m-vector, s, of Lower Bounds on Optimal Basestock levels

1. $\forall i$
2. $\lambda_i = \Sigma_{j \ni i}\, a_{ji} * \lambda_j$

3. $p_i = \Sigma_{j=\{i\}} \frac{\lambda^j}{\lambda^i} + \Sigma_{j \ni i, j \neq \{i\}} \frac{\lambda^j}{2 \max_{c \epsilon j}\{\lambda_c\}} (p^j + \sum_{c \epsilon j, c \neq i} h_c)$

4. $\forall i$

5. $s_i =$ ComputeStochasticBasestock(λ_i, l_i, p_i, h_i)

7 Search Algorithms

We implement search algorithms to obtain solutions better than those given by closed-form approximations. [9] suggest that the a global optimal solution can be reached in polynomial time using a Steepest Descent or Best Improvement method. This claim, however, rested on the ability to evaluate the cost function $C(s)$ exactly. They rely on existing methods for minimization of submodular functions to find the best neighbor policy. The simulation-based Best Improvement search relies on an exponential number of simulations to find the minimum of the neighborhood that they suggest, and is thus impractical, despite its it optimal performance. We use the same methods in generating bounds to the optimal policy but use simulation-based comparisons and experiment with search algorithms that will be more practical than Best Improvement.

This section will describe the general structure of the algorithm we use to find a basestock, and several searches that we have implemented and tested.

Algorithm: FindCorrelatedPolicy
Inputs: ATO System, StartingBound, SearchType, SimDays
Outputs: Vector s of component basestock levels
Initialize:
UpperBound $\leftarrow$ ComputeUpperBound(ATO)
LowerBound $\leftarrow$ ComputeLowerBound(ATO)
$s \leftarrow$ StartingBound
$s' \leftarrow s$

1. while ($s' = s$)
2. $s' \leftarrow s$
3. $s \leftarrow$Search(ATO system, s, SimDays)
4. Return s

To initialize, we call the ComputeUpperBound and ComputeLowerBound algorithms from above. We experiment by altering the search used in step 3. The "Steepest Descent" search was implemented as follows:

Best Improvement Search
Inputs: ATO System, s, SimDays
Output: s^*
Initialize:
$s^* \leftarrow s$
$H \leftarrow \{s' \mid \forall i, (s_i - s'_i) \epsilon \{0, 1\}\}$

1. $\forall s' \epsilon H$
2. if Evaluate(s', ATO, SimDays) < Evaluate(s^*, ATO, SimDays)
3. $s^* \leftarrow s'$
4. return s^*

To reduce search time, we replaced Best Improvement Search with First Improvement Search, or "Hill-Climbing":

First Improvement Search
Inputs: ATO System, s, SimDays
Output: s^*
Initialize:
$s^* \leftarrow s$
$H \leftarrow \{s' \text{st.} \| (s_i - s'_i) \| = 1, i \epsilon \{1 \dots m\} \wedge s_i = s'_i, h \neq i\}$

1. while $(s^* = s \wedge H \neq \emptyset)$
2. randomly choose $s' \epsilon H$
3. $H \leftarrow H - s'$
4. if Evaluate(s', ATO, SimDays) < Evaluate(s^*, ATO, SimDays)
5. $s^* \leftarrow s'$
6. return s^*

Since First-Improvement finds a local minimum (as does steepest descent, though with a greater probability that the local will also be a global minimum, in our case), the initialization is important. Typically, the search can be executed multiple times, with random restart positions used. We tried starting it twice, using each bound as a starting position. Finally, we improved on the performance of First Improvement by implementing Simulated Annealing:

Simulated Annealing
Inputs: ATO System, s, SimDays, Cooling Schedule
Output: s^*
Initialize:
$s^* \leftarrow s$
$s' \leftarrow s$ $H \leftarrow \{s' \text{st.} \| (s_i - s'_i) \| = 1, i \epsilon \{1 \dots m\} \wedge s_i = s'_i, h \neq i\}$
T according to Cooling Schedule

1. while $(s^* = s \wedge H \neq \emptyset)$
2. randomly choose $s'' \epsilon H$
3. $H \leftarrow H - s''$

4. $\triangle(s', s'') \leftarrow$ Evaluate(s'',ATO,SimDays) - Evaluate(s',ATO,SimDays)
5. if rand $[0, 1] \leq \exp\{-\frac{\triangle(s',s'')}{T}\}$
6. if Evaluate(s'', ATO, SimDays) < Evaluate(s^*, ATO, SimDays)
7. $s^* \leftarrow s''$
8. Update T according to schedule
9. return s^*

For a cooling schedule we obtained good results by initializing $T = 4$, and cooling according to the schedule $T_i = 4 * .8^{\frac{i}{2}}$, where i is the total number of search iterations that have been executed. Each iteration consists of a full execution of the algorithm above, which itself is called iteratively from FindCorrelatedPolicy. All of these searches are iterative and generate a new neighborhood for their solution until a local a minimum is found, though this part of the search has been abstracted into the FindCorrelatedPolicy algorithm in this section.

8 Simulation

Our search is dependent on our ability (imperfect as it may be) to evaluate the level of costs that we expect to incur when implementing a particular basestock policy, and thus to choose the better of two or more policies. We do this by simulation, using our own ATO simulator. We simulate both the behavior of the market, through randomly generated demands and leadtimes, and the behavior of our agent, by managing inventory with an Order-Up-To mechanism.
The design of the simulator is as follows:

ATO Simulator
Input: ATO, Simulation Length, Policy
Output: Cost/Day

- The product structure of the ATO is expressed as an $n \times m$ matrix A, where a_{ij} is the number of units of components i needed to assemble one unit of product j.
- For each product, there is a sale price and a demand distribution associated with it.
- Each day's demand is generated by sampling n Poisson demand distributions.
- For each component, there is a cost of purchase and a leadtime distribution associated with it.
- Leadtime is discretely distributed, with an expected delivery date, and probabilities associated to a finite number of delay lengths. A component's leadtime distribution is randomly sampled every time an order is placed for the component, to determine when that components will arrive (this information is stored internally in the order and only the distribution is known to the agent).

A high-level view of a simulation day is as follows:

1. Supplier Progress: All orders due are moved one day closer to arrival. Orders that have reached their arrival date are removed from the outstanding orders queue and added to the stock of physical inventory. When a component arrives, its cost is subtracted from the starting balance.

2. Daily Demand: All of the product demand distributions are sampled, and the matrix A is used to convert $d^j(t)$ into $d^j(t) * A = d_i(t)$. Component demands are then added into the queue of backorders waiting to be filled. Existing backorders and the newly generated demand are filled until supply runs out, at which point, a late penalty is assessed for all items remaining in the backorder queue. When an item is filled, its sale price is added to the balance.

3. Restock: Orders are placed for components according to the policy that was input at the start of the simulation. Specifically, for each component i, $z_i = s - y_i + w_i - x_i$ is ordered, and that quantity is added to the queue of orders due from the supplier. A leadtime, sampled from the distribution is assigned to each order.

4. End of Day: A holding penalty is subtracted from the balance for each unit of stock that remains in physical inventory at the end of the day. In TAC, interest will be accumulated at this time, with holding costs coming in the form of lost interest.

Since we are trying to minimize computation time, we want to keep our simulations as short as possible. However, a large number of cycles are required for convergence to steady-state values and thus for accurate cost predictions. Still, the policy that performs better in a short simulation is likely, with some likely probability, to perform better in the steady-state. Part of our research has been directed at determining a simulation length that will evaluate correctly with a large probability while using only a minimum of computation time. The table below shows the average distance form the mean in samples of cost estimates taken at different lengths in an ATO system that we studied. The variance continues to reduce dramatically as the simulation length is extended up to 20,000 days of simulation, but stabilizes thereafter. Thus, we accept the results of 25,000 day simulations as steady-state results and base on out performance evaluations on these figures. Simulations performed while searching must be considerably shorter, with a goal of simulating less than 100 days. The table shows a good deal of variance in the cost estimates at this length. There is enough variance that the policy chosen through a comparison based on data obtained at this length will not necessarily be the better policy with steady-state data. The effects of this error on the performance of policies obtained through search are examined in the Results section.

Simulations Days	Avg Distance from Mean in 10 samples
5000	4.215259999999989
10000	4.260071999999968
15000	3.844680000000005
20000	3.7984373333333226
25000	1.9263360000000376
30000	1.79210000455

Figures 2-4 in the appendix show data collected from simulations of a two-product system with correlation. Multiple polices were simulated in an attempt to characterize the cost function for the system.

- Figure 2 shows data collected from 25,000 day simulations with policy increments of 5 to show the general convexity of the cost function.

- Figure 3 shows data collected from 25,000 day simulation with policy increments of 1. The curve is not smooth, because their is correlation, but there is only a global minimum.
- Figure 4 shows data collected from 15,000 day simulations, and stochastic leadtimes (Figure 2 used deterministic, which speeds convergence). The curve has a similar general shape, but local minima exist. Actual searches will have more variance than in figure 4.

9 Results

The following is a summary of results we obtained in testing of search algorithms on a number of two and three-product ATO's. We tested over a range of simulation lengths (for evaluation) and cost ratios. Refer to the appendix for more specifics. For consistency, we present data collected, except when it is stated that we are testing a range of a particular variable, from the following three-product ATO, which we will call the "Test System".

ATO: TestSystem
Components: A,B,C
$\forall i, \varphi(l_i)$:

- 3 days: 50%
- 4 days: 30%
- 5 days: 20%

Each component has a cost of $\$10$ per unit.
Holding costs were identical for each components but varied among simulations.
Products: A,B,ABC

- Product A: $50 per unit, p_A = 30, Poisson Demand $\lambda = 3$
- Product B: $90 per unit, p_B = 54, Poisson Demand $\lambda = 6$
- Product ABC: $90 per unit, p_{ABC} = 54, Poisson Demand $\lambda = 6$

9.1 Closed-Form Performance

One of the most important results we obtained was that the bounds proposed by [9] worked well in practice. In experiments conducted on two and three product systems with correlated demands, we consistently found the optimal to lie between these bounds. Moreover, the bounds were sufficiently tight to suggest that they may be of use as approximations in environments where decision time is severely limited; TAC being among these situations.

Though tests on larger systems are still necessary, these results are encouraging. In comparison of the performance results of the upper bound policy to those of the optimal policy, we consistently found the costs/day to be within 5% of the optimal in 25000 day simulations.

9.2 Computation Time

Using the upper bound as a policy would have obvious advantages in terms of computation time, as no search is necessary. Even among searches, there was a significant range of search times. Since the limiting factor in the speed of a search is the number of simulations that are needed before termination of the algorithm, we compared computation time by keeping track of the average number of simulations needed by each algorithm as an indicator.

Search procedures were tested on the Test System, with a range of simulation lengths. Below are average number of simulations per search for 200 searches spread uniformly over the range of simulation lengths $\{10, 20, \ldots, 100\}$

- Steepest Descent: 22.2 sims
- First Improvement: 6.2 sims
- First Improvement w/ restart: 11.4 sims

Clearly, there is a computational advantage for either First-Improvement algorithm in comparison to Steepest Descent. Since Steepest Descent must evaluate every policy in its neighborhood, the number of simulations it requires makes it unattractive for use in a TAC environment. Finally, the advantage gained by First Improvement was so drastic, that restarts could be used while maintaining an improved search time. This improved performance over the single First Improvement considerably.

9.3 Search Performance

The improved search time of the First-Improvement search did not come without a tradeoff in performance. Both Steepest Descent and First Improvement find global optimum policies given perfect evaluation, and both fell short of this mark when using simulation data. First Improvement, however, following an often more convoluted path to the optimal, was more susceptible to errant evaluations. Thus, the Steepest Descent begins consistently near-optimal behavior at much shorter simulation lengths than First Improvement. (See Figure 5)

This affect carried through into the shortest simulation lengths, where Steepest Descent outperformed First Improvement in all but those cases where the optimal was extremely close to one of the bounds. However, First Improvement w/ Restart and Simulated Annealing performed significantly better, challenging the average performance of Steepest Descent in considerably less time.

9.4 Conditions

Since the performance of First Improvement was overly sensitive to starting position, "Restarts" seem to be a necessary extension. By searching from both bounds, First Improvement w/ Restart was less affected by changes in the cost ratio, whereas First Improvement search from the upper or the lower performed well only under certain conditions (compare performance of "firstupper" and "firstlower" in Figure 7 and Figure 8). Steepest Descent also proved to be relatively robust. Results are given for a range of holding costs (with penalty cost held constants) to gauge the effect that the cost ratio had on each policy's effectiveness.

9.5 Conclusion

Other than Steepest Descent, all of the algorithms seem to be heavily affect by the starting location at low simulation lengths. Still, the time saved is enough that several restarts can be executed in the time of a single Steepest Descent algorithm. It is clear, that in the case of small simulation lengths, a global minimum is far from assured. Under these conditions, a greedy algorithm is likely not to be the best method since it cannot recover from a mistake made as as result of error in evaluation. This explains the performance gains made by Simulated Annealing relative to First Improvement. Unfortunately, (from an efficiency viewpoint), the more evaluations that are made, the lower the probability is that the overall performance will be significantly affected by errant values. This would suggest that it is worth the time to try a longer search. However, the accuracy of our closed form approximations, and perhaps our ability to add further heuristics about the policy to which we should initialize, make it fairly certain that using multiple randomized searches, that are likely to terminate quickly, will turn out a good result.(See Figure 6)

10 Future Work

10.1 Heuristics

The success of closed form approximations suggests that it will be possible to perform reasonably well without spending any time searching. The experiments showed that higher ratios of holding to shortage costs yielded optimal policies closer to the lower bound,and lower ratios yielded policies nearer the upper bound. A future problem will be to determine a heuristic strategy that bids the upper or lower bound depending on some type of average cost ratio. Preliminary results using the average of upper and lower bounds, as well as a weighted average toward the (generally) tighter upper bound have been promising.

10.2 Leadtime Correlation

Demand correlation is a difficult problem for which some background exists, but leadtime correlation is a very much untouched topic. We are attracted to this problem (of correlation across time periods) because of closeness to real life situations, where a supplier delay in one time period is likely to indicate a shortage that could “ripple” into future periods. This problem is difficult in that it violates many of the key assumptions of our model, namely the applicability of a queuing system, which ensures steady-state convergence in the model that we have been using.

10.3 Leadtime Decisions

Another way in which the leadtime could be more accurately modeled is by changing leadtime to an endogenous variable, taking only cost/leadtime tradeoffs as exogenous. This is representative of the TAC environment, as well as real life situations to which SCM agents may be applied.

[1.]Kenneth Arrow, Samuel Karlin, and Herbert Scarf. “Studies in the Mathematical Theory of Inventory and Production” California: Stanford University Press (1958)

[2.]Guillermo Gallego, "Material Requirements Planning (MRP) - IEOR 4000 - Production Management"
Citeseer url: citeseer.nj.nec.com/493501.html

[3.]Hillier and Lieberman, “Introduction to Operations Research”, 7^{th} edition, New York: McGraw-Hill Textbook form AM120

[4.]H. Van Dyke Parunak, Robert Savit, Rick L. Riolo, Steven J. Clark, “DASCh: Dynamic Analysis of Supply Chains” (1997)
Citeseer url: citeseer.nj.nec.com/parunak99dasch.html

[5.]Fikri Karaesmen, Yves Dallery “A Performance Comparison Of Pull Type Control Mechanisms For Multi-Stage Manufacturing” (1998)
Citeseer url:citeseer.nj.nec.com/karaesmen98performance.html

[6.]Fikri Karaesmen, John A. Buzacott , “Integrating Advance Information In Pull Type Control Mechanisms For Multi-Stage Production”
Citeseer url : citeseer.nj.nec.com/204724.html

[7.]Lee YH, Kim SH. “Optimal Production-Distribution Planning in Supply Chain Management using a hybrid Simulation-Analytic Approach” (2000)
url: http://www.informs-cs.org/wsc00papers/169.PDF

[8.]Yingdong Lu, Jing-Sheng Song, David D. Yao “Order Fill Rate, Leadtime Variability, and Advance Demand Information in an Assemble-to-Order System” (2001)
Citeseer url: citeseer.nj.nec.com/484612.html

[9.]Yingdong Lu, Jing-Sheng Song, David D. Yao "Order-Based Cost Optimization in Assemble-to-Order Systems". (2002)

[10.]Murota, Kazuo. "L-Convex Functions And M-Convex Functions". (1998)
Citeseer url = citeseer.nj.nec.com/murota98lconvex.html

[11.]Murota, Kazuo. "Algorithms in Discrete Convex Analysis". TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems (2000)
Citeseer url = citeseer.nj.nec.com/murota00algorithms.html

[12.]Jing-Sheng Song, Paul Zipkin, "Assemble-to-Order Systems". Chapter 13 in *Handbooks in Operations Research and Management Science,* VOl. XXX: *SupplyChainManagement*. T.de KoK and S.Grave(eds.). North: Holland
url = http://www.gsm.uci.edu/ song/Working%20Paper/Atocto03Nov02.pdf

[13.]Viswanadham Srinivasa, "Lead Time Models for Analysis of Supply Chain Networks"
Citeseer url : citeseer.nj.nec.com/208334.html

[14.]Ramesh Srinivasan, Rangarajan Jayaraman, James A. Rappold, Robin O. Roundy, Sridhar Tayur “Procurement of Common Components in a Stochastic Environment” (1998)
Citeseer url : "citeseer.nj.nec.com/srinivasan98procurement.html"

[15.]Sridhar Tayur “Computing Optimal Stock Levels for Common Components in an Assembly System” (1995)
Citeseer url =:‘citeseer.nj.nec.com/159591.html’

[16.]Daniel Dajun Zeng, Katia Sycara. “Dynamic Supply Chain Structuring for Electronic Commerce Among Agents” (1999)
Citeseer url : citeseer.nj.nec.com/zeng99dynamic.html

[17.]Wagner, HM, and TM Whitin, “Dynamic Version of the Economic Lot Size Model” *Management Science*, Volume 5, Issue 1 (Oct. 1958), 89-96

Figure 1: Single-Item Example (Section 3.3.1)

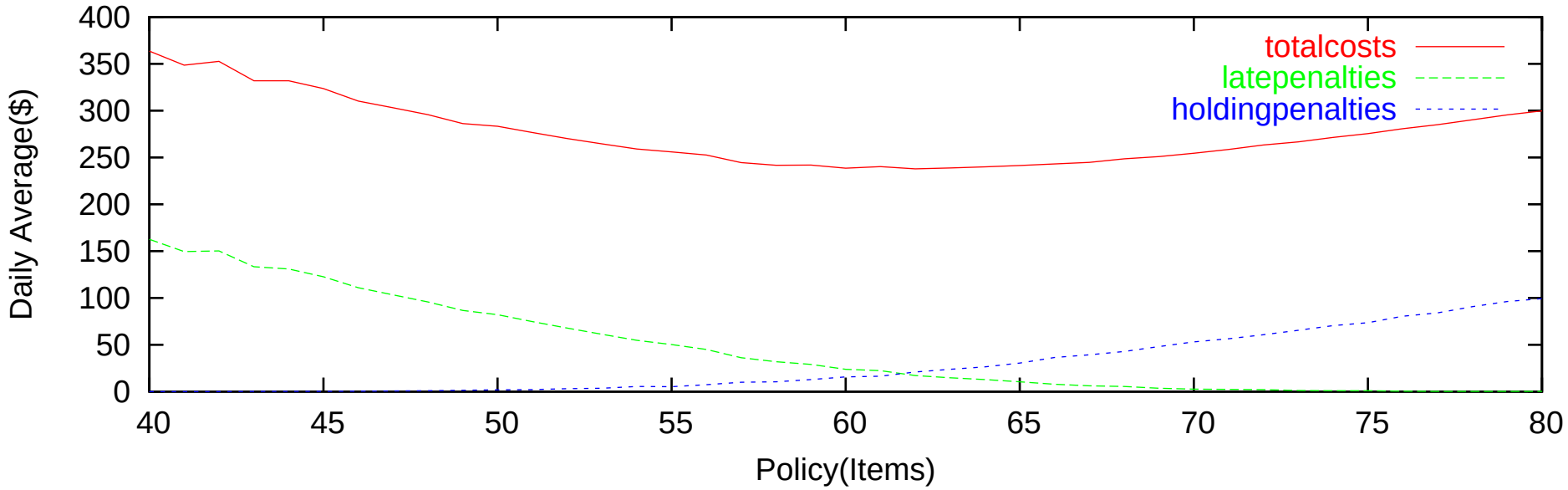

Figure 1: Single-Item Example (Section 3.3.1)

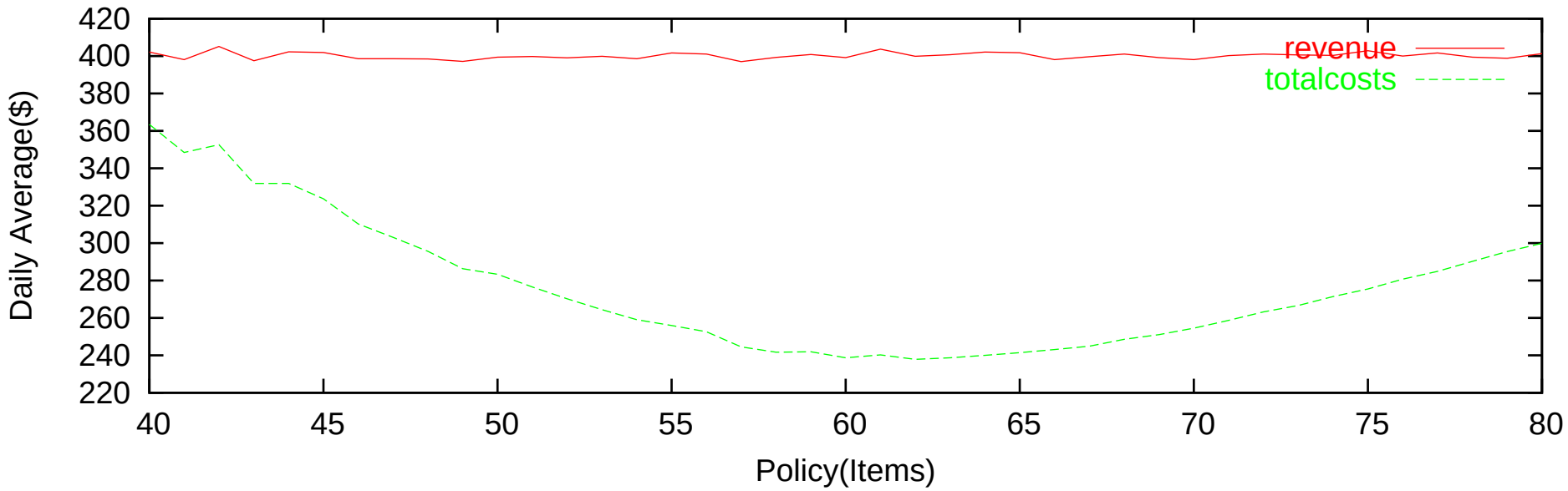

Figure 2: Cost/Day A:20, AB:20

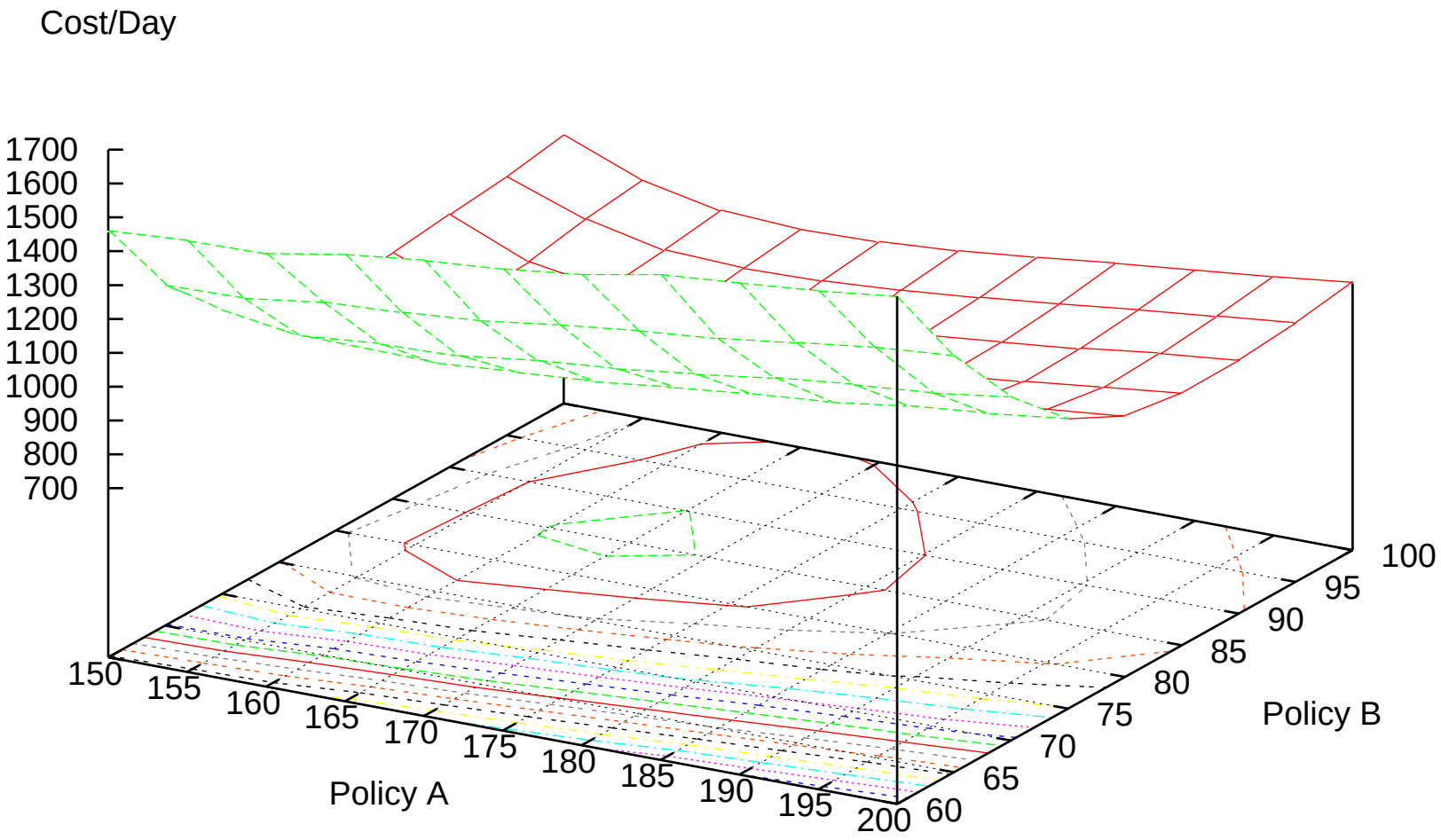

Figure 3: Cost/Day A:20, AB:20 detLT

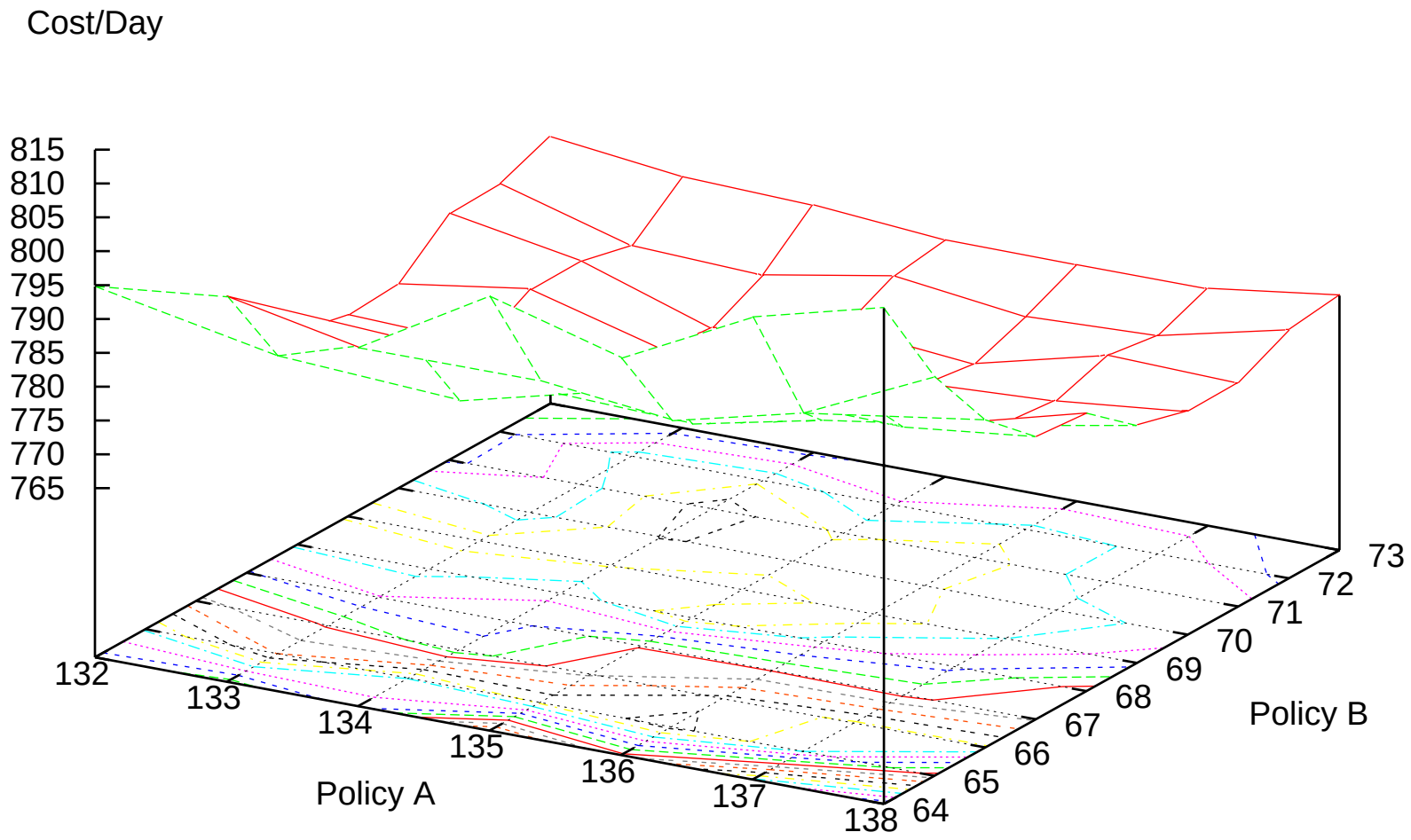

Figure 4: Cost/Day A:20, AB:20

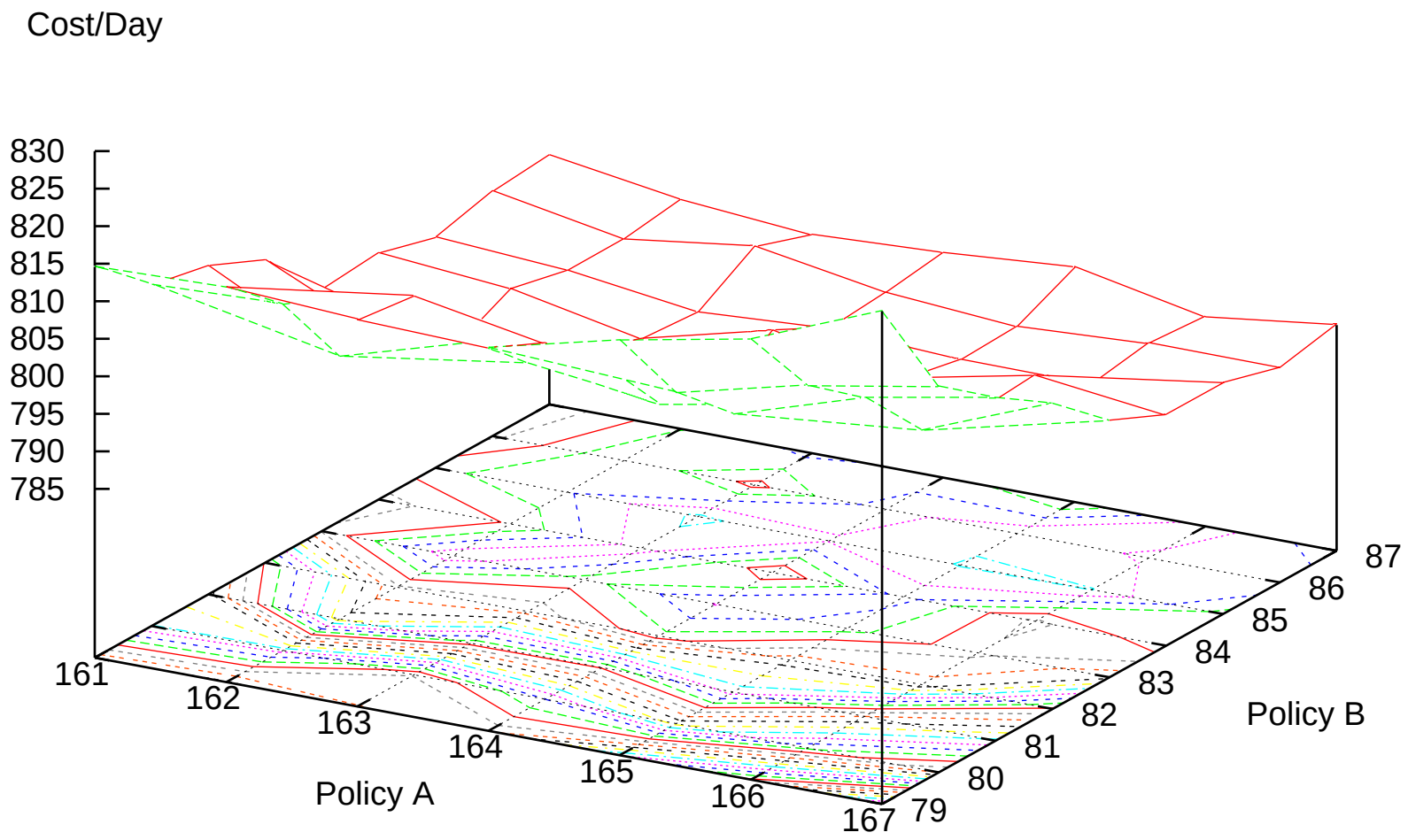

Figure 5: Performance of Longer Searches in 3-product setting with h = 20

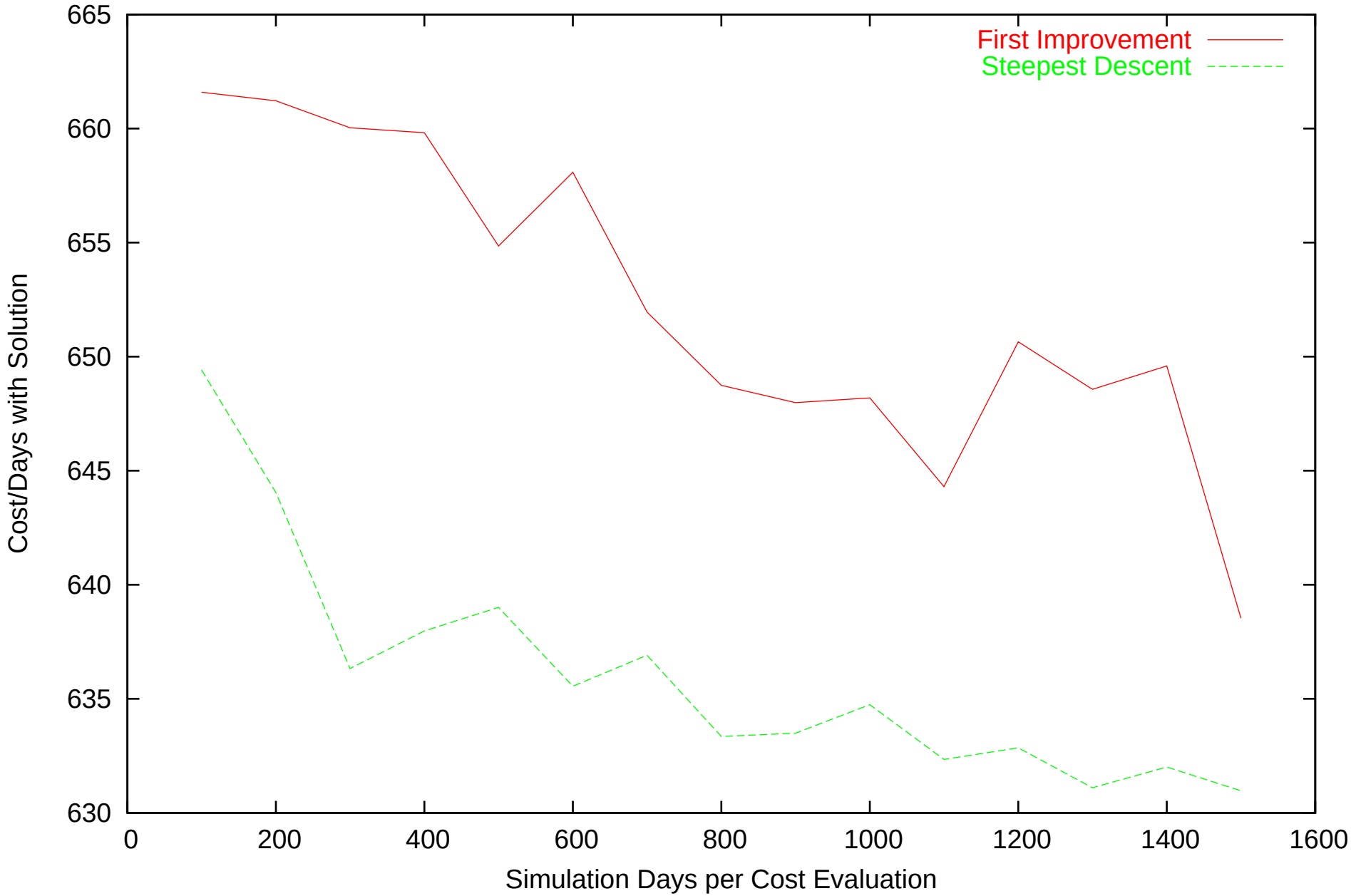

Figure 6: Effects of Cost Ratio on Performance

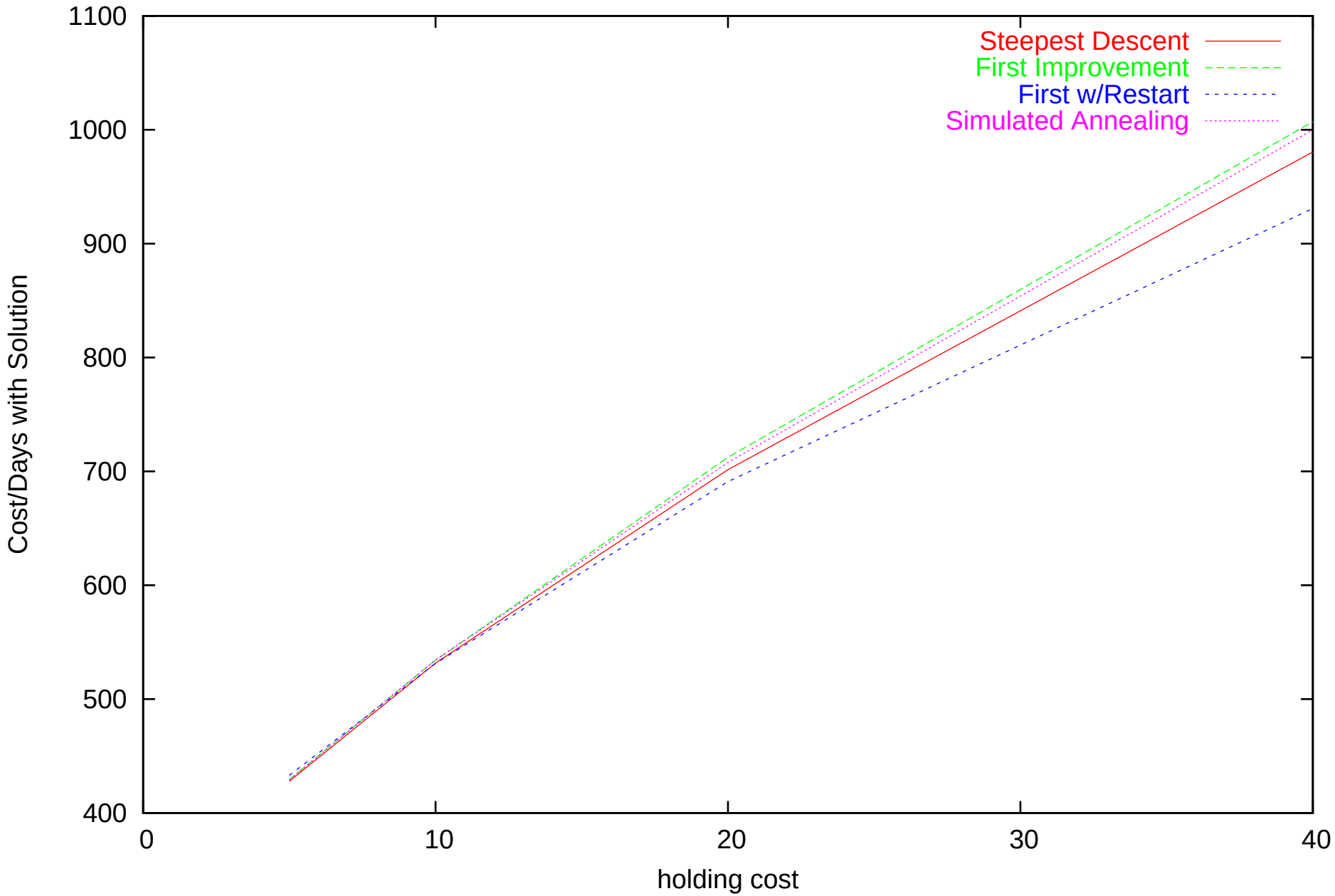

Figure 7: Performance of Search Solutions for three product ATO, h = 20

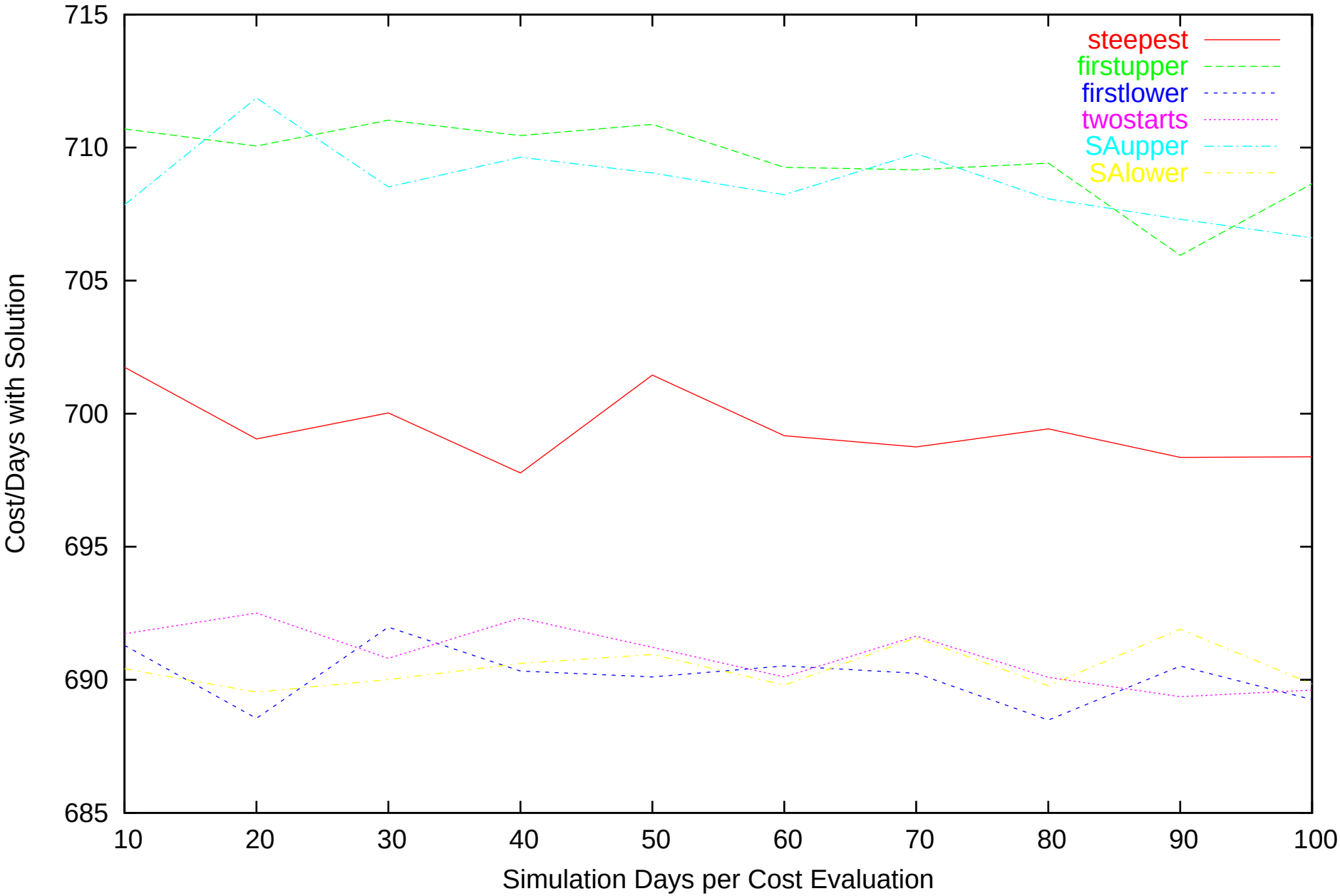

Figure 8: Performance of Search Solutions for three product ATO, h = 5