

A Simulation Study of Warehouse Storage Assignments

By

Victoria M. Eddy

A Thesis
Submitted to the Faculty
of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the
Degree of Master of Science
in
Industrial Mathematics

April 2004

APPROVED:

Professor Carlos Morales, Thesis Advisor

Dr. Bogdan Vernescu, Head of Department

Contents

1	Introduction	6
2	A Linear Programming Approach	9
2.1	The Simplex Method	9
2.2	The Simplex Method Applied to the Warehouse Storage Problem	14
3	Problem Description	14
3.0.1	Product Flow	14
3.0.2	Reserve Rack Description	14
3.0.3	Pick and Drop Locations	16
3.0.4	Product Transportation Means	16
3.0.5	Problem Formulation and Results	18
3.0.6	Notation	18
3.0.7	Storage Costs	19
3.0.8	Retrieval Costs	20
3.0.9	Designating Pick and Drop Locations (P&D)	21
4	A Simulation Study of Storage Assignment Policies	26
4.1	Basics of the Simulations	26
4.1.1	The Cost Matrix	26
4.1.2	The Occupancy Matrix	27
4.1.3	Available Locations Cost Matrix	27
4.1.4	Product Type Matrix	28
4.1.5	Generating Tickets	30
4.1.6	Inventory and Demand	32
4.1.7	Completing Retrieval Tasks	32
4.1.8	Differences in Policy Simulations	32
4.2	Results	33
5	Simulation Code:	42
5.1	Turnover Based Simulation Code:	42
5.2	Current Policy Simulatin Code	46

List of Figures

1	This diagram shows the main features of the actual warehouse.	15
2	Random Policy: The bottom figure is a depiction of the cost matrix using in the simulations. The top figure is a color coded matrix of types of items depicting the initial state of the Type matrix. Note that highly demanded items (in dark blue) are in the most expensive locations). The center figure depicts the final state of the Type matrix.	37
3	Random Policy: This graph shows a sample path of the total daily cost (in blue). Note the sharp decline in cost as times passes by. This decline is due to an <i>improved</i> Type matrix via the policy. The red line depicts the minimum daily cost under the optimal allocation of items (i.e. under the unrealistic assumption that tickets are completely predictable).	38
4	Turn Over Policy: The two top graphs show the initial and final Type matrices in the simulation. The cost matrix appears at the bottom.	39
5	The blue line shows the daily cost for a simulated path of tickets. The red line is the cost for the optimal placement of items (under the unrealistic assumption that tickets are completely predictable).	40
6	Random vs. Turn Over Comparison: This graph show the percentage loss in cost-wise efficiency for both the Random Policy and the Turn Over based one over the optimal policy. Each line depicts the ratio of the realized cost under the respective policy over the cost under the optimal policy.	41

Acknowledgments

I would like to express my warm thanks to my advisor, Professor Carlos Morales, for his kindness and patience in helping with this project. I would also like to thank the Mathematics department, and the students and faculty who have helped me throughout my studies at WPI. I have learned so much in such a short amount of time here. It has truly been a challenging, but rewarding experience that I shall carry with me forever.

I would also like to thank my colleagues at Staples for supplying me with data and knowledge of internal operations. Another thank you, to my undergraduate professors from Saint Anselm College: Gregory R. Buck, Peter W. Lindstrom, and Donald L. Stancl, who helped me to discover my love for mathematics, while giving me the courage and confidence to take my studies to the next level.

Most of all, I would like to thank all of my friends and family for their unlimited support and faith. A special thank you to my parents: Keith Eddy and Lillian and Joseph O'Rourke, who have been there for me every step of the way with their moral and financial support. Without them, I would be lost. Another special thank you to Stacy Walker, who has been by my side even during my hardest moments, so that I have never felt alone.

Abstract

Warehouse operations have become an important part of the retail industry today. For many companies in the retail industry, the profit margin can be as little as one cent for every dollar sold. Because of these extremely small profit margins, it is important that companies save as much in costs along the way as they can. One such way is to cut down on warehouse costs. In this paper, we shall study the costs of storing and retrieving products in a pick, pack and ship warehouse for an office supply company. In particular, we will examine possible ways to improve the flow of products into and out of reserve racking, by implementing different storage assignment policies..

One way of finding the best possible storage assignments would be to formulate a cost equation along with constraints, and to minimize the cost using linear programming techniques. We shall study the Simplex Method, and we will use it to find optimal locations for a given set of storage tasks. However, the set of products to be stored and retrieved in our warehouse changes from day to day, and is subject to customer demand. Because there is no closed form solution or algorithm to find the optimal storage assignment policy under a random situation, we will use simulation techniques to study different storage assignment policies that could be applied in the warehouse. We will study the efficiency along with the maintenance requirements for each of the different policies and compare them with the current policy being used in the warehouse today.

1 Introduction

The goal of this project is to study possible ways to improve storage and retrieval of products within a warehouse. This particular warehouse contains a pick, pack, and ship operation for office supply products. In this warehouse, shipping cartons flow around a system of conveyors leading into and out of modules. Inside the modules, office supplies are manually picked and placed into the cartons. A section of reserve racking in the warehouse serves as the supplier of products to the picking modules. It stores products received by the warehouse until they are needed to replenish the picking modules. The products that are to be retrieved from storage each day are printed on tickets which are compiled over night. Tickets are lists of items that are to be moved from their storage locations to the picking modules. The products selected depend upon the demand of customers, who place their orders the day before either by telephone or online. The order in which products appear on a ticket is entirely determined by customer orders. Another way of explaining this is that, the greater the demand of a product, the more likely it is to be needed in the picking module, and therefore the more likely it is to show up on the ticket for a particular day. However, where the product appears on the ticket depends on what time of the day the customer made his or her order. The tickets printed are usually different from day to day.

In this project, we would like to study possible ways to improve the flow of products into and out of the reserve racking. One way of solving this type of problem would be to formulate a cost equation, and minimize the cost by determining the optimal storage assignments, using linear programming techniques. In this project we will discuss a linear programming technique, the Simplex Method, and apply it to a sample problem. However, the inputs/tickets of the warehouse storage problem involve a great deal of randomness. Since there is no closed form solution or algorithm to find the optimal storage assignment policy under a random situation, we shall study a second approach using simulation that allows us to deal with the factor of randomness.

Results will be achieved by implementing storage policies, that impose constraints on the selection of open locations for incoming loads of product (Van Der Berg & Gademann, 2000). Three such policies studied by researchers in automated storage and retrieval are *random assignment*, *turnover based assignment*, and *class based assignment*. Random assignment allows a product to be stored anywhere in the reserve rack, where as a turnover

based assignment policy places more frequently needed products in storage locations associated with lower costs(in terms of time), and less frequently used products in locations associated with higher costs (Van Der Berg & Gademann, 2000). Class based assignment distributes products, based on their demand rates, into classes, for which regions within reserve racking are assigned (Vanden Der Berg and Gademann, 2000). For example, classes of product with higher demand will be assigned to storage zones associated with lower travel times, where as classes of product with lower demand will be assigned to zones with higher travel times. Within the classes and corresponding zones, products are randomly assigned. If the assigned zone for a particular product is full, the next most costly zone will be used since placing slower moving products into lower cost zones fills up and occupies these zones, preventing their use by faster moving products (Van Den Berr & Gademann, 2000).

The three previously mentioned storage assignment policies were first compared by Hausmann et al (1976) and then by Thonemann & Brandeau (1988). In their studies, turnover based assignment showed marked improvement in storage and retrieval time over random assignment. Class based assignment also outperformed random assignment, but did not realize all of the savings of turnover based assignment. Along with varying degrees in performance, these policies also vary in the degree of requirements necessary to maintain them. Turnover based assignment requires that the demand of each individual product be measured and compared to all other products. Class based assignment relieves this burden by grouping products into classes, so that only the classes must be compared as a whole, rather than the individual products. Random assignment of course has the least amount of investment since it does not require knowledge of product demand or location costs. The policy currently in use by the warehouse in consideration is a type of random assignment. In this policy, products are placed in the best available location as they come in, without regard to demand or turnover of the product.

In this project, we will study and compare the currently used policy and the turnover based policy, and use this analysis help determine the most appropriate assignment policy to be used in the reserve racking of our pick and pack warehouse. We shall see that when applied to the given conditions of our warehouse, in the long run a turnover based policy yields no major improvements in daily costs compared to the currently used policy. However, when started from a situation in which products are placed in

the most unfavorable positions (ie: high demand products in high cost locations), the turnover based assignment policy lowers costs close to optimal costs, significantly quicker than the current policy.

2 A Linear Programming Approach

2.1 The Simplex Method

Given a set of products to store and retrieve and the respective costs of storing and retrieving these products, it is possible to optimize the placement of products into storage in the form of a linear programming problem.

Definition 1 *A linear programming problem (LP) is an optimization problem for which we do the following:*

1. *We attempt to maximize (or minimize) a linear function of decision variables. The function to be maximized*

(or minimized) is called the objective function.

2. *The values of the decision variables must satisfy a set of constraints, which are in the form of linear equations or linear inequalities.*

2. *A sign restriction is associated with each variable. For any variable x_i , the sign restriction specifies either that x_i*

must be non-negative or that x_i be unrestricted in sign. (Winston, 1987)

In our case, the objective function is a cost function expressing the total cost of storing and retrieving the given sets of products in a ticket. The decision variables will be the variables X_{ij} and Z_{ij} , each indicating respectively whether or not the given product i will be stored in location j , and whether or not product i will be retrieved from location j . The restrictions will be specified in the requirements of the storage and retrieval.

Example 2 *For example, one requirement that we will encounter is that only one product may reside in a given location at a time. This*

means that we cannot assign more than one product to a single location.

We would satisfy this constraint with the

inequality:

$$\sum_i (X_{ij}) \leq 1 \forall j$$

This inequality says that if we sum all indicator variables of products going to a particular location j , the sum is less than or

equal to one. Since $X_{ij} \geq 0$, we can only put up to one pallet of products into each location.

We will be using the Simplex Method via the Excel Solver to solve the cost equation of our sample problem. Before we go on, let us first try to gain an understanding of the Simplex Method and to see how it works on a simple LP problem.

As described by Wayne L. Winston in *Operations Research: Applications and Algorithms*:

Definition 3 *Any basic solution to a system $Ax = b$ of m linear equations and n variables (assume $n \geq m$) in which all variables are non-negative is called a **basic feasible solution**.*

Theorem 4 *The feasible region for any linear programming problem is a convex set. Also, if an LP has an optimal solution, there must be an extreme point of the feasible region that is optimal.*

Theorem 5 *For any LP there is a unique extreme point of the LP's feasible region corresponding to each basic feasible solution. Also, there is at least one basic feasible solution corresponding to each extreme point of the feasible region.*

These two theorems tell us that the extreme points of an LP's feasible region are the basic feasible solutions for the system. In searching for a minimal solution to an LP we need only find the basic feasible solutions corresponding to the smallest cost equation value (Winston, 1987).

We shall next demonstrate the Simplex Algorithm by walking through the simple example of solving the minimization problem:

$$\min z = 4x_1 - x_2$$

Under the constraints:

$$\begin{aligned} 2x_1 + x_2 &\leq 8 \\ x_2 &\leq 6 \\ x_1 - x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \end{aligned}$$

The feasible region for our LP is quadrilateral formed by the x-axis, the y-axis, the line $y = 6$, and the line $y = 8 - 2x$. This is the only area that satisfies all of the constraints simultaneously. We know from the theorems above that the solution must be at one of the extreme points of this region.

The Simplex Algorithm has four basic steps:

Step One: Convert the LP to standard form.

Definition 6 *An LP is in standard form when it is converted into an equivalent problem in which all constraints are equations and all variables are non-negative. (Winston, 1987)*

The standard form for our LP is:

$$\min z = 4x_1 - x_2$$

such that:

$$\begin{aligned} 2x_1 + x_2 + s_1 &= 8 \\ x_2 + s_2 &= 6 \\ x_1 - x_2 + s_3 &= 4 \\ x_1, x_2, x_3, s_1, s_2, s_3 &\geq 0 \end{aligned}$$

Definition 7 *The s_i 's are **slack variables** representing the amount of resource unused in the i th constraint. Slack variables must always be greater than or equal to zero. (Winston, 1987)*

We add these slack variables so that we can change the inequalities of the constraints to equalities.

Step Two: Find a Basic Feasible Solution

Since all the constraints are \leq constraints with non-negative right-hand sides, the slack variables may be used as the basic variables for each row (Winston, 1987): Therefore, the basic variables are: $BV = \{z, s_1, s_2, s_3\}$,

and the non-basic variables are $NBV = \{x_1, x\}$. The basic feasible solution is:

$$z = 0, s_1 = 8, s_2 = 6, s_3 = 4, x_1 = x_2 = 0$$

Row	z	x_1	x_2	s_1	s_2	s_3	rhs	Basic Var.
row 0	1	-4	1	0	0	0	0	$z = 0$
row 1	0	2	1	1	0	0	8	$s_1 = 8$
row 2	0	0	1	0	1	0	6	$s_2 = 6$
row 3	0	1	-1	0	0	1	4	$s_3 = 4$

Step Three: Determine if the BFS is Optimal/Determine the Entering Variable

When minimizing an LP, a bfs is optimal if all non-basic variables in row zero have non-positive coefficients. In our case, the variable x_2 has a positive coefficient. We take the non-basic variable with the most positive coefficient and enter it into the basis. When entering a variable into the basis, we use the ratio test to determine which row to enter the variable into.

Definition 8 *In the ratio test, we compute the ratio of the right hand side of the row divided by the coefficient of the entering variable for each row in which the entering variable has a positive coefficient. $\frac{rhs}{coefficient}$. The largest ratio is the value of the entering variable that will keep the current basic variables non-positive.*

x_2 is positive in both rows one and two. We compute the ratio test for both rows as follows:

$$\begin{aligned} \text{row 1: } & s_1 \geq 0 \text{ for } x_2 = \frac{8}{1} = 8 \\ \text{row 2: } & s_2 \geq 0 \text{ for } x_2 = \frac{6}{1} = 6 \end{aligned}$$

Therefore, row 2 is the winner of the ratio test.

Step Four: Perform row operations to make the entering variable the basic variable in any row that wins the ratio test.

We enter x_2 into the basis at row 2 with a coefficient of one, and perform row operations so that x_2 has a coefficient of zero in the remaining rows 0,1 and 3:

To create a coefficient of 1 for x_2 in row 2, we simply multiply row two by 1(since the coefficient is already 1).

To create a coefficient of 0 for x_2 in row 0, we set

$$\text{row 0} = -(\text{row 2}) + (\text{row 0})$$

To create a coefficient of 0 for x_2 in row 1, we set

$$\text{row 1} = -(\text{row 2}) + (\text{row 1})$$

To create a coefficient of 0 for x_2 in row 3, we set

$$\text{row 3} = (\text{row 2}) + (\text{row 3}).$$

The resulting table is:

Row	z	x_1	x_2	s_1	s_2	s_3	rhs	Basic Var.
row 0	1	-4	0	0	-1	0	-6	$z = -6$
row 1	0	2	0	1	-1	0	2	$s_1 = 2$
row 2	0	0	1	0	1	0	6	$x_2 = 6$
row 3	0	1	0	0	1	1	10	$s_3 = 10$

Now that we have a new canonical form, we return to step 3, using the current canonical form.

Definition 9 *A system of linear equations is said to be in canonical form when each equation has a variable with a coefficient of one in one equation, and a coefficient of zero in every other equation.*

The basic variables are now $BV = \{z, s_1, x_2, s_3\}$, and the non-basic variables are $NBV = \{x_1, s_2, s_3\}$. The bfs is now:

$$z = -6, s_1 = 2, x_2 = 6, s_3 = 10$$

Now we see that all non-basic variables in row zero have non-positive coefficients, meaning that the solution is optimal with

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix} \text{ and } \min z = 4x_1 - x_2 = 4(0) - 6 = -6$$

2.2 The Simplex Method Applied to the Warehouse Storage Problem

Now that we have seen how the Simplex Algorithm works, we shall apply it to solve a problem involving our warehouse storage assignments. The problem will be solved using the Simplex Algorithm via Excel Solver. We shall first describe the operation in detail, followed by the formulation of the problem, including notation and calculation of storage and retrieval costs. Finally, given a set of products to be stored and a set of available locations, we will use the simplex algorithm to optimize the assignment of the products.

3 Problem Description

3.0.1 Product Flow

The flow within the warehouse is from receiving to reserve rack, and then as needed, from reserve rack to the picking modules.

3.0.2 Reserve Rack Description

The reserve racking area consists of seven aisles with racking on both sides and the front aisle, which only has one side of racking, facing the picking module. The racking is seven levels high, with 62 bays per row. Racking positions are designated by; aisle: bay: level, with even numbered bays on one side of the aisle and odd numbered bays on the other side. For instance 31-01-01 indicates aisle 31, bay 1, level 1, and 31-02-01 indicates the storage location directly across from it. For the purposes of this project, travel costs for even and odd bay counterparts are assumed to be equal since travel paths to and from these locations will be the same. Each bay has two compartments for storage, also assumed to have equal travel costs.. The warehouse under consideration has two sections of reserve racking separated by a center aisle. The Eastern section of racking is labeled section 1 and the western section 2. The picking module is at the northern end of the building and runs parallel to the two sections of reserve racking. The receiving area is on the eastern side of the reserve racking. (See diagram)

3.0.3 Pick and Drop Locations

Products enter and exit the reserve racking from Pick and Drop Locations (P&D). P&D are located at the eastern end of each row in section 1. These P&D are essentially in the receiving area and are referred to as receiving P&D. There are also P&D locations at the end of each row of both sections in the center aisle, and these are referred to as center aisle P&D. Entrance and Exit P&D's are the same physical locations. They are called entrance P&D when products are placed there until they are moved into the reserve rack location. They are called exit P&D's when products are placed there that have been retrieved from the storage location already and are waiting to be taken to the picking module.

3.0.4 Product Transportation Means

A turret machine retrieves products from entrance P&D on pallets and brings them to their respective storage locations. When products are needed from reserve racking, a turret machine retrieves them from their location in storage and drops them off at their respective exit P&D's. An order selector truck picks up products from their exit P&D and puts them away in their proper location in the picking module. What this translates into essentially is that: If looking at the diagram, turret machines travel horizontally (across the reserve racking aisles) and order selectors are responsible for vertical travel (up and down the center aisle and the receiving aisle). Double pallet jacks move product to center aisle P&D when these P&D are used for entrance.

Equipment Speeds and Capabilities

Order Selector: An order selector truck is used to move product from exit P&D to the picking module. It travels at a speed of 5.7mph or 8.36 ft/sec. It is a very mobile truck and can move freely through aisles without the use of a grooved track. The truck can raise and lower a pallet to different levels of racking however the aisles of reserve racking in the warehouse are too narrow for the order selector to angle its forks in position to retrieve product. Therefore the order selector cannot be used to put product into or take it out of reserve racking.

Height	MPH	FT/Sec
0 - 60	5.7	8.36
61 - 120	3.8	5.57
121 - 150	2.9	4.25
151 - 180	2.0	2.93
181 - 366	1.2	1.76

Turret Truck The turret truck is used to move products from entrance P&D to reserve rack locations, and to move products from reserve rack locations to exit P&D. It is a less mobile piece of equipment than the selector truck. It travels at a maximum speed of 6mph or 8.8 ft/sec, however it must travel on a grooved line that helps to guide it up and down the aisles. Therefore, a turret truck is very efficient when traveling up and down an aisle, but it is much slower in changing aisles and operating off its track. The turret truck has special forks that raise and lower to different levels. The forks rotate allowing the turret truck to retrieve product in the narrow reserve rack aisles. The turret truck can travel while raising and lowering its forks, but it does so at slower speeds. The turret can raise and lower its forks at 1.28 feet per second. The horizontal speeds at which the turret can travel, while the forks are at different levels, are listed below.

Level Zone	MPH	FT/Sec
0 - 17.5'	6.0	8.8
17.5 - 23.33'	4.2	7.04
23.33 - 29.17'	2.6	3.52
29.17 - 40.81'	1.0	1.47

Double Pallet Jack: The double pallet jack is a smaller piece of equipment used to move pallets within the receiving area and to P&D locations. It can carry two pallets at a time at various speeds depending on the weight of the load. It can only move products at ground level and does not have the capabilities to raise products or retrieve them from higher levels. In this study, we will use 8.95 FT/Sec as the pallet jack speed since the typical pallet of product will not exceed 4000 lbs.

Weight	MPH	FT/Sec
Empty	7.5	11.0
4000 lbs.	6.1	8.95
6000 lbs.	5.7	8.36
8000 lbs.	5.3	7.77

3.0.5 Problem Formulation and Results

For this part of the project, the costs for each location in the reserve rack will be determined in terms of travel time from receiving to the storage location (storage costs) and then from the storage location to the picking module (retrieval costs). Given a number of products to be stored, we shall select a set of locations, with varying storage and retrieval costs, that we will assume to be available. A cost equation will be formulated along with a given set of constraints. The cost equation, constraints, and the given sets of products and available locations will then be input into Excel Solver to optimize the storage assignments for the products. We are assuming the currently used assignment policy in which products are stored in the best available location in the order of which they are received. Therefore, we shall not take into consideration the demand/turnover of the product.

3.0.6 Notation

- M Denotes the number of products to be moved from receiving to storage locations in reserve racking.
- S Denotes the number of products to be moved from storage locations to the picking module.
- N Denotes the number of storage locations in the reserve racking area.
- C_j Indicates the cost of moving a product from receiving to storage location j .
- P_j Indicates the cost of moving a product from storage location j to the picking module.
- $X_{(i,j)}$ Is an indicator variable with $X_{(i,j)} = 1$ if product i moves to location j and $X_{(i,j)} = 0$ if product i is not moved to location j .

- $Z_{(i,j)}$ Is an indicator variable with $Z_{(i,j)} = 1$ if product i moves from storage location j to the picking module and
 $Z_{(i,j)} = 0$ if product i is not moved from storage location j to the picking module.

3.0.7 Storage Costs

Storage costs are dependent only upon the distance that products must travel, and the speeds of equipment being used. Costs are independent of the type of product being moved. The distance to storage consists of the East/West distance traveled by the turret machine from the receiving end of the aisle to the location. The exception is when center aisle entrance P&D are used for section one, in which case the pallet jack time must be added (See *Designating Pick and Drop Locations* below for further detail). This is half the time it takes for the double pallet jack to bring product from receiving to the designated center aisle P&D. We use half the time, since the pallet jack is transporting two pallets of product at a time.

Because the turret travels at slower speeds while raising or lowering its forks, a change level time and distance is calculated for each level (1 thru 7). The change level time is calculated by dividing the distance the forks travel in each level zone by the speed at which the forks raise/lower. This time is then multiplied by the horizontal speed at which the turret can travel while in the level zone to get the horizontal distance traveled while in the level zone. The table below displays the time required to raise the forks to the various levels(or to lower forks to the ground from the indicated level). The third column indicates the total distance traveled while the truck is raising(or lowering) its forks based on the speeds that it can travel while the forks are at different heights. For example, it takes the turret 17.89 seconds to raise its forks to level five. During this time, the turret truck travels a horizontal distance of 149.49 feet.

Level	Lower/Raise Fork Time	Distance Traveled While Raising/Lowering Forks
1	0	0 feet
2	4.47 seconds	39.29 feet
3	8.95 seconds	78.72 feet
4	13.42 seconds	118.05 feet
5	17.89 seconds	149.49 feet
6	22.36 seconds	165.20 feet
7	26.83 seconds	171.77 feet

The total storage cost is the time it takes the turret to reach the designated storage location from the receiving end of the aisle plus half of the double pallet jack time, if there is any. This is equivalent to the combination of the total change level time, plus the time the turret travels while its forks are at ground level. If the turret reaches the designated storage location before the forks reach the correct level, the turret is no longer traveling horizontally, but the entire change level time must be applied until the forks are raised to the level of the location.

3.0.8 Retrieval Costs

Retrieval costs consist of the time it takes the turret to travel from the storage location to the closest P&D plus the time it takes the order selector to carry the product from the P&D to the picking module. The turret time is the same as explained in storage costs except that now the forks are being lowered rather than raised. The same change level distances and times apply since the forks raise and lower at the same speed. The order selector does not raise or lower its forks during its trip from the exit P&D to the picking module, so it travels at the same speed throughout. Therefore, its costs are simply the distance from the P&D to the picking module, multiplied by the speed at which it travels. Travel time for the order selector starts at the exit P&D, and the order selector can go up the center aisle or the aisle next to the receiving area depending on which P&D it is starting from. If it is starting from a center aisle P&D then it will travel up the center aisle, and if it is starting in a receiving P&D then it will travel up the aisle in receiving.

3.0.9 Designating Pick and Drop Locations (P&D)

Entrance P&D When a product is assigned to a location in reserve racking, it must first be placed at a P&D location from which the turret machine can pick it up. Product must be placed at a P&D that is at the end of the aisle into which it will be stored. This is done because the turret machines are guided by grooved lines in the rows. By placing products in the P&D at the end of the row to which it must enter, we prevent the turret from having to leave its track. For product going to section two, only the center aisle P&D may be used. However, for section one, product may be placed at P&D on either side of the designated aisle. We assign zero cost for placing product on P&D that are in the receiving area, but we must calculate a cost for transporting product from receiving to the center aisle P&D locations. This transporting is done by a double pallet jack, which can take two pallets (two products) around the northern side of the reserve rack area and down the center aisle to the designated P&D. Because products may be moved in two's to the center aisle P&D, it is sometimes cheaper (uses less travel time) to use center aisle P&D to put products in locations of section 1 that are close to the center aisle. For example, for two locations close to the center aisle, using the receiving P&D requires that a turret truck travels two long round trips from receiving to the storage locations, while using center aisle entrance P&D only requires one long trip by a pallet jack, and two short trips by the turret truck.

In the warehouse management system (WMS), for each location in reserve racking, we designate a P&D location that will be used to enter the row when a product is to be stored in this location. There are no options for assigning section 2 P&D's since they are only located at one end of the row. However, section 1 locations may use P&D's at either end of the row. To optimize the assignment of these P&D's, for each location we first calculate the times to store two products using the center aisle P&D and the receiving side P&D. For the receiving side P&D we have zero cost in transporting the two products to the P&D plus the cost of turret machine traveling from the P&D to the location and back twice (since we are moving two products, one at a time). For the center aisle P&D we have the cost of the pallet jack moving two products from receiving to the P&D plus the cost of the turret moving the product from the P&D to the location and back twice. It may seem as though the receiving side P&D would always be cheaper since it has no cost for moving products to the P&D itself. However, for locations in close to the

center aisle it is actually cheaper to incur this added cost in order to shorten the trips of the turret machine. Therefore we designate P&D's for section 1 based on which one is cheaper to move two products. In the excel diagram P&D Designations, the shaded cells indicate locations in which using center aisle P&D's is cheaper. WMS requires that P&D locations are assigned in whole sections of racking. In other words, we cannot separate the different levels in racking. For example, if we assign bay 119 level one to a specific P&D, we must also assign levels 2,3,4,5,6 and 7 of bay 119 to that same P&D.

The values in the cells of the diagram indicate the savings by using the center aisle P&D for that location. The shading indicates that it would be best to block off the Western end of section one rows for center aisle P&D use. We must now determine the best bay number at which to block each row off. To do this we start at the lowest numbered bay that is shaded. We block this bay and every bay to the left of it off, and add all of the cells in the blocked off section to determine the savings in time. We then move to the next highest bay number and repeat the procedure. We do this for the remaining bays. The section with the highest sum contains the locations for which we will designate the center aisle P&D (see P&D Designation). All other locations in the row will use the receiving P&D.

Exit P&D Section 2 locations again have no choice but to use the center aisle P&D for exit. To optimize section 1 location assignments we want to find the lowest cost for a turret machine to move a product from the location to a P&D. This is of course done by taking the shortest distance from the location to the nearest P&D. Therefore, each location in section 1 is assigned to the exit P&D closest to it. For example, locations on western side of a row use the center aisle P&D for exit, and locations on Eastern side of a row use the receiving P&D for exit.

Simplex Method Applied to Modified Problem In this simplified version of the warehouse storage problem there are M products to be put into storage. The M products will be placed into N storage locations with $M \leq N$. We will then assume that the M products are demanded, and therefore all M products will be retrieved from storage and transported to the picking module.

Some assumptions of the operation are that:

- Costs of storage and retrieval are as described above: They equal the cost of transporting a product from receiving to the proper storage location, plus the cost of transporting the product from the storage location to the picking module. Entrance and exit P&D are assigned based on the criteria described above.
- All machines transport only one palleted product at a time, except for the double pallet jack, which carries two palleted products at a time.
- Each storage location may contain a maximum of one palleted product.

Constraints:

Using the notation described earlier, the Cost equation is:

$$LP = \sum_{(i,j)} (C_j X_{(i,j)} + P_j Z_{(i,j)})$$

-

To ensure that each product has a location we must have:

$$\sum_j (X_{(i,j)}) = 1, \forall i$$

- To ensure that each location has no more than one product:

$$\sum_i (X_{(i,j)}) \leq 1, \forall j.$$

- All products must be stored:

$$\sum_{i,j} (X_{(i,j)}) = M = \sum_i m_i.$$

- All products must be retrieved:

$$\sum_{i,j} (Z_{(i,j)}) = S = \sum_j s_j.$$

For this simplified problem we shall consider only three products ($M = 3$) and five possible storage locations ($N = 5$). We assume that we are also retrieving all products from storage and transporting them from the module ($S = 3$) For the sake of simplicity, we shall label the products, 1, 2, and 3, and the storage locations 1, 2, 3, 4, and 5. X_{ij} will be the indicator variable for product i being stored in location j . X is a variable matrix of zeroes and ones. Z_{ij} will be the indicator variable for product i being retrieved from location j . Z is a variable matrix of zeroes and ones also. C_{ij} and P_{ij} are the costs respectively for storing product i in location j , and subsequently for retrieving product i from storage location j and transporting it to the module.

- Using the above constraints, we now have:

- $\sum_{j=1}^{j=5} X_{ij} = 1 \forall i \implies$ The sum of each row in X must be one. This constraint insures that each of the three products has one and only one possible location.
- $\sum_{i=1}^{i=3} (X_{ij}) \leq 1 \forall j \implies$ The sum of each column in X must be less than or equal to one. This constraint insures that no more than one product may be placed in each storage.
- $\sum_{i=1}^{i=3} \sum_{j=1}^{j=5} (X_{ij}) = M = 3 \implies$ The sum of all of the elements in X must equal the number of products to be stored, ensuring that all products are stored in the reserve racking.

The assumption in this problem is that all products that are stored are also retrieved and transported to the module. For this we must have $Z = X$. All constraints of X apply to Z , the cost equation simplifies to:

–

$$\begin{aligned}
 LP &= \sum_{i=1}^{i=3} \sum_{j=1}^{j=5} (C_j X_{ij} + P_j Z_{ij}) \\
 &= \sum_{i=1}^{i=3} \sum_{j=1}^{j=5} (X_{ij} (C_j + P_j))
 \end{aligned}$$

Solver Results

- – The given cost matrix is

$$C + P = \begin{matrix} & \begin{matrix} 11.95 & 77.14 & 14.32 & 75.60 & 12.95 \end{matrix} \\ \begin{matrix} 11.95 & 77.14 & 14.32 & 75.60 & 12.95 \\ 11.95 & 77.14 & 14.32 & 75.60 & 12.95 \end{matrix} & \end{matrix}$$

$$\text{and the results: } X = \begin{matrix} & \begin{matrix} 0 & 0 & 1 & 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{matrix} & \end{matrix}$$

$$LP = 1(14.32) + 1(12.95) + 1(11.95) = 39.22$$

The solution is obvious in that the algorithm placed the three objects to be stored in the cheapest available locations while following the occupancy constraints. This same algorithm can be useful

when trying to complete storage assignments for vast quantities of products and locations. When trying to apply this approach to a modification in which only a subset of the stored products are actually retrieved, a more powerful tool than Excel Solver is needed. A second matrix of ones and zeroes is necessary to indicate which products are retrieved, and this exceeds the number of adjustable cells in Excel Solver, which is 200.

4 A Simulation Study of Storage Assignment Policies

Now that we have explored how we could solve a deterministic form of the problem we shall discuss a different approach that will deal with the randomness involved in the creation of storage and retrieval tasks. Using simulation, we would like to study how different policies perform under a random situation. In this section, we will describe in detail the implementation of the different simulations. The simulations are very similar except for in the ways that they each one replenishes the reserve racking. We will first discuss the basics of the simulations which are common among all. Then we will follow with the details behind each storage assignment policy and explain how these details change the replenishment aspect of the simulation. Finally, the results of the simulations will be discussed, along with recommendations for which policies would best serve the warehouse under consideration.

4.1 Basics of the Simulations

The simulations of the different storage assignment policies all share a majority of their key elements. All of the elements discussed in this section are exactly the same in each of the simulations. Also embedded in the description of each of the elements will be the discussion of some of the general assumptions made about the operations of the warehouse for the purpose of this study.

Storage and retrieval tasks will be performed on a daily basis, for a specified number of days D . A ticket, formulated at the start of the day, will list the retrieval tasks for that day. The number of items on the ticket is set to 30% of the total number of locations. This percentage is chosen to be similar to the actual daily turnover rate for this section of racking in the warehouse.

4.1.1 The Cost Matrix

In all of the simulations we must associate storage and retrieval costs for each of the possible locations. In the simulations we consider one section of reserve racking from our pick, pack and ship warehouse. This section is seven levels high and 60 bays long, yielding 420 possible storage locations. The costs for these locations were taken from the formulation of the LP problem

discussed early in thesis. In the beginning of each simulation we introduce a 7×60 cost matrix $Cost$. Each element in the cost matrix corresponds to the added storage and retrieval cost for using that location. For instance, the value of the cost matrix $Cost_{(i,j)}$ contains the storage plus retrieval cost for the location in level i and bay j . The cost matrix remains constant throughout simulation.

4.1.2 The Occupancy Matrix

An "occupancy matrix," X , is initialized early on in the simulation to keep track of which locations in the racking are empty or filled. We consider a location empty if it has no product, and filled if one pallet of product has been placed there. Not more than one pallet of product may be allowed in one location at a time. The occupancy matrix takes on the dimensions of the cost matrix, with an element by element correspondence. X is a matrix of ones and zeroes with $X_{(i,j)} = 0$ if the location in level i and bay j is occupied, $X_{(i,j)} = 1$ if the location is empty. In the initialization of X we make all of the locations full so that each element of X is a zero. As we retrieve products from locations in the reserve racking, we update X by changing the zeroes to ones as the locations become empty. Then as we replenish the racking by placing newly arriving products into empty locations, we change the corresponding ones to zeroes to indicate the appropriate occupancy and so on. The section of racking under consideration is in a prime section of reserve area, meaning that it contains locations with some of the cheapest storage costs. Because of this, the racking is almost always close to 100% occupancy, and most of the time locations fill up in a short amount of time upon vacancy. For these reasons, the location is initialized at full occupancy. While retrieval tasks are being performed, the racking will show vacancies of up to 70%, but after replenishment tasks are completed for each day, it will be returned to full occupancy.

4.1.3 Available Locations Cost Matrix

Throughout simulation, "an available locations cost matrix" AC will be used to indicate the costs for only those locations that are empty. AC is equal to the dot product of the occupancy matrix and the cost matrix ($X \cdot Cost$). Because all occupied locations in X are zeroes, when we take the dot product we get a new matrix with zeroes in all the locations that are full. The

elements in AC corresponding to empty locations, will contain the combined storage and retrieval cost for that location, since $X_{(i,j)} = 1$ in all empty locations. This matrix will primarily be used during replenishment.

4.1.4 Product Type Matrix

Along with keeping track of the costs and occupancy of each location, it would also be of value to know what type of product is being stored in each location. The office supply warehouse that we are considering has over 5,000 different products from pens and pencils to computers, printers, and scanners. However, not all of these items have the same amount of demand among customers. For example, the company's brand name copy paper has the highest demand of other product we sell. While on the other end of the spectrum, we have products that can go through periods of time where there are barely any sales at all. To capture this aspect in the simulation, we introduce types of products. It would be too difficult to keep track of 5,000 different products in our simulation, so we narrow it down to 20 types of products, with each one of the locations in racking containing a pallet of one these 20 types. The types are labeled by the numbers from 1 to 20 as listed in the chart below. Also listed below are the demands. These demands and product types presented below have been fabricated to protect the integrity of the company. However, the demand percentages used are similar in magnitude and variance to real demands inside of the warehouse.

Product	Type	Demand
Company Brand Paper	1	.101
Other Brands of Copy Paper	2	.1
High Gloss Paper	3	.094
Pens	4	.09
Desks	5	.09
Ink Cartridges	6	.075
Binders	7	.07
Printers	8	.05
Printers w/ Copier & Fax	9	.05
Wireless Routers	10	.05
Fax Paper	11	.03
Folders	12	.03
Scientific Calculators	13	.02
Calculators	14	.02
Staplers	15	.02
Erasers	16	.02
Fax Machines	17	.02
Chairs	18	.01
Calenders	19	.01
Phones	20	.01

A "type" matrix called *Type* is created and initialized by using Monte Carlo sampling as follows:

- First we create a matrix of zeroes and ones by drawing a number between zero and one, from a uniform distribution, $\text{Uniform}(0,1)$.

Next, we create an array (*cp*) of the cumulative sums of the proportion of sales for each type. We will illustrate this method with an example *cp* of length ten. The actual *cp* is for twenty products and therefore is of length twenty-one.

Example 10 *Example 11*

$$cp = [.36 \ .388 \ .4 \ .52 \ .6 \ .6715 \ .73 \ .818 \ .84 \ 1]$$

Next we append a zero to the front of this matrix:

Example 12

$$\begin{aligned} cp &= [0 \quad cp] \\ &= [0 \ .36 \ .388 \ .4 \ .52 \ .6 \ .6715 \ .73 \ .818 \ .84 \ 1] \end{aligned}$$

Next we cycle through each element of random numbers in the matrix *Type* looking for values between the first and second numbers (0, .36) in the array *cp*. For each of these values we change the element of *Type* to a 1, indicating that this location in reserve racking contains a type 1 product. Next we cycle through *cp* again, this time looking for elements of *Type* with values between the second and third numbers in the array *cp*. For each of these values, we change the element of *Type* to a 2, indicating that corresponding storage location contains a type 2 product. We continue this process through the interval between the last two elements in *cp* (.84, 1) for which we change all elements of *Type* with values in this interval, to 10, indicating that the corresponding storage locations contain type 10 products.

Now we have a matrix *Type* of the same dimensions of the occupancy matrix *X* and the cost matrix *Cost*. Each element in *Type* is a number from 1 to 20 corresponding to the type of product contained in the location, and the number of each type of product in storage has been randomly chosen according to its proportion of sales, by means of a Monte Carlo sampling. For example, $Type_{(i,j)} = 5$ indicates that the storage location, in level *i* and bay *j*, contains a product of type 5. Now, whenever we update the occupancy matrix *X* as items are either put into or taken out of storage locations, we also update the type matrix *Type*. We do this by entering the type of product being stored in $Type_{(i,j)}$ as we change $X_{(i,j)}$ from 1 to 0, and by entering a zero in $Type_{(i,j)}$ as we change $X_{(i,j)}$ from 0 to 1, as products are removed. If we are removing a product, we insert a zero in the location of the type matrix to indicate that there is no product in that location. At the end of each day in the simulation, a snapshot of *Type* is taken so that we can observe any patterns of where the different types of products are being stored over time.

4.1.5 Generating Tickets

A ticket is generated before the start of each day. This ticket contains the retrieval tasks, or the list of items to be retrieved from storage and transported to the picking module. The ticket is generated to reflect customer demand

of products. We do this with a Monte Carlo sampling on the Multinomial distribution taken from (Winston L, 1987). The function that creates the tickets in our simulation is called `mutlirnd`. This function takes as inputs, the number of items to be retrieved from storage T and the vector p of the demands of the product types 1 thru 20.

$p = [.101 .10 .094 .09 .09 .075 .07 .05 .05 .05 .03 .03 .02 .02 .02 .02 .02 .01 .01 .01]$

We create a vector m for which $m(1) = \text{binornd}(T, p(1))$
(The length of m is L .)

The first element of m is the total success of T Bernoulli trials with the probability of success $p(1)$.

$m(i) = \text{binornd}(T - \text{sum}(m(1 : i)), \frac{p(i)}{\sum_i^L P(i)})$ for $1 < i < L$

These values of m are the number of success in the $T - \text{sum}(m(1 : i))$ Bernoulli trials, with the probability of success $\frac{p(i)}{\sum_i^L P(i)}$.

If the sum of the elements of m reaches the total number of items to be placed on a ticket before we reach the end of m , then the remaining elements of m are set to zero.

Now we have a vector m for which $m(i)$ is the number of products of type i to be placed on the ticket. If $T - \sum_{i=1}^{i=L} m(i) > 0$, then $m(L) = T - \sum_{i=1}^{i=L} m(i)$.

We create an array *Ticket* with $m(1)$ one's, $m(2)$ two's, ..., $m(10)$ tens. The length of *Ticket* is of course T . We randomize the order of the ticket by aligning the elements of *Ticket* with a permutation of the numbers from one to T , and putting the elements of T in the positions of the permutation.

Example 13 *If we have a ticket of length 5, with three types of products:*

$Ticket = [1 \quad 1 \quad 2 \quad 2 \quad 3]$

We create a permutation of the numbers 1 to 5

$Permutation = [5 \quad 1 \quad 3 \quad 4 \quad 2]$

Now we set the elements of Ticket in the new positions corresponding the number it lines up with in permutation.

$Ticket = [1 \quad 3 \quad 2 \quad 2 \quad 1]$

The order of the ticket is made random to account for the randomness in which orders are placed by customers and received by customer service.

4.1.6 Inventory and Demand

An important aspect of warehousing is keeping track of what a building actually has, and what it needs to meet the demand. The scope of this project is not focused on determining proper levels of inventory, but we must keep track of what we have and what we need in case a ticket is created that asks for more than what we have stored in our rack. For this reason, we create two vectors *Inventory* and *Demand*. The *Inventory* vector contains the total of each type of product that we have in storage. The *Demand* lists the total number of each product that the daily retrieval ticket is asking for.

4.1.7 Completing Retrieval Tasks

Retrieval tasks are completed in order that they appear on the ticket. The simulation first looks at the task and searches the type matrix for all of the locations that contain this type of product. If the product cannot be found because we have run out, it places the item on the end of a "MissingItem" vector that keeps track of every item that is skipped because the inventory cannot meet the demand. Once the simulation finds the list of locations that contain the type we want, it looks through the cost matrix and finds all of the costs of these locations. It then retrieves the product from the location with the minimum cost. If there are multiple locations with the same cost, it takes the first one on the list. Each time an item is pulled, the occupancy and type matrices are updated accordingly.

4.1.8 Differences in Policy Simulations

Now that we have explained everything that the simulations have in common, we shall now look at the differences. There are two different policies that we are simulating. The only difference in the simulations is the way in which we replenish items. These differences as well as the policies, are discussed below.

Current Policy Replenishment In the policy that we are currently running in our facility, we do not keep track of the demand of products, but

rather we place them back into storage in order of which we receive them. Therefore, for any set of storage tasks, the tasks are completed in order, with the item in the first task being placed in the best available location, the second item on the list being placed in the next best location, and so on.

The storage tasks are generated in a special way to simulate the circumstances of the warehouse. When a product is running low in the warehouse, an order is placed to the vendor of that product. However, the arrival of the ordered product varies depending on how far away the vendor is, and what time of day they ship their goods. This means that the storage tasks are not necessarily in the same order as the retrieval tasks. In the simulation, every product that is pulled out of storage is replenished before the start of the next day. However, to simulate how the facility really runs, the order in which the storage tasks appear is made random. Everything that is retrieved on a particular day, is placed on a receipt. The order of the receipt is then made random and is then submitted as the list of storage tasks for the day. This way the inventory is kept the same for the start of the next day, to prevent inventory issues in the simulation.

Turnover Based Replenishment In turnover based replenishment, products on a list of storage tasks are put away with the items of the highest demand stored in the best locations and the items of lowest demand in the most costly locations. To simulate this in our study, we take a list of storage tasks and place the items in order of descending demand. They are then replenished one at a time.

4.2 Results

The two policies discussed above were run in two sets of simulations. Simulations were run with 20 product types over a period of 500 days, which totals to about two years of operation when we consider only work days. It was expected that the turnover based policy would show better results than the current policy:

Results Sim 1	Current Policy	Turnover Based Policy
Total Cost	1422308	1069967
Daily Cost	2844	2139
Daily Cost Std. Dev.	54.326	176.85

The results at first seemed to show that the Turnover Based Policy yielded better results in terms of overall costs and daily costs. We expected that the turnover based approach would start out at a higher average daily cost and make its way to a lower average daily cost. However neither policy showed significant improvements over time. The higher standard deviation in the turnover based policy raised question about the starting points of the two simulations. It was believed that this might have been caused by starting the simulations with different initial type matrices. The type matrices for the first set of simulations were randomly generated. If the starting point of one policy was more favorable then the other, this would skew the results. It was decided that a better approach would be to start both simulations from the same type matrix.

Because of the results of the first set of simulations, the second set of simulations were started from the same type matrix. This time the turnover based policy now had higher cost than the current policy. These results gave us no real proof of savings in switching from the current policy to the turnover based policy under the given set of conditions in our facility. The costs of both policies seemed to already be in a steady state, with small variations in daily cost, but niether significantly better than the other.

Rate of Convergence to Steady State We performed another set of simulations in order to asses how fast the two afformentioned replenishment policies reach a steady state. By steady state, we mean the state when costs do not seem to improve over time, and in fact the daily cost seems to follow a random walk. In order to see how fast steady state is reached under each policy, we start a simulation from a very *bad* initial matrix (a Type matrix with the worst placement of items). As tickets are generated, we observe the behavior of cost improvement under each policy, with the turnover based policy reaching the steady state in about half the time as the current policy.

In investigating why there seemed to be no advantage to applying turnover based storage assignments to a randomly generated Type matrix, we realized that we were actually starting the simulation at a point where it had already reached a steady state. The daily costs were no longer showing significant improvement nor differentiation from each other. Had the randomly generated type matrix been a more unfavorable one, in which a significant number of products with high turnover were placed in locations associated with high costs, we would have seen that the daily costs of both policies drop over time,

but that the turnover based policy would do so in a much shorter period of time.

To illustrate the reduction of costs from the initial "bad" matrix, we graphed the daily costs for each policy over time along with the optimal cost for each day. The optimal cost was considered to be the cost if all products to be retrieved that day were in the lowest cost locations possible. As time progresses in the simulation, the costs for both policies approach the optimal cost, with the turnover policy always a little closer. However, the advantage or savings of the turnover based policy becomes less significant with time. Because the policies yield relatively the same results in the long run, it would not be cost efficient to switch from the current policy to the turnover based policy in our warehouse, since doing so would require very costly system updates so that arriving products could be recognized and associated with a demand.

Further studies should be conducted to see if treating products as individuals, rather than grouping them into types, would yield better results. Also, a study should be considered in which both simulations are started from an "optimal" type matrix in which all products are placed in locations according to demand. Theoretically, the turnover based policy should be in an absorbing state in which the type matrix is the same every day, since no further sorting can be done. Where as the current policy will continue to experience fluctuations in cost, since products are put back in random order.

A Markov Chain Interpretation Until now, we have assumed that the simulation strategy will show how cost behaves under the current and turn over policy. However, results from this simulations are valid only if the policies generate sequences of cost that are in some sense stationary. In order to prove that the costs sequences from each of these policies is stationary, we can make use of powerful theorems from the theory of Markov Chains.

A Markov Chain is a sequence of random variables with the property that conditional on the present, the future is independent of the past. In the context of the present situation, we can think of the sequence of Type matrices as they change daily as a random sequence of matrices. At any point in time, the probability that a Type matrix will take on a particular configuration depends solely on the previous day's configuration (and today's ticket). Hence, conditional on today's configuration, one can figure out the probability that the Type matrix will take on any other configuration tomorrow, solely based

on information known today. As a matter of fact, since tickets are all independent and identically distributed, the *transition probabilities* for a Type matrix to change its state depend only on the current state, and are the same over time. Therefore, the sequence of Type matrices in our simulations form a Markov Chain.

That the Type matrix is a Markov Chain is not sufficient for our results to carry a sense of validity. We would like this Markov Chain to lead to a stationary distribution for the Type matrix. In order to show that the Type matrix sequence will indeed converge to a stationary sequence, one would have to prove that the Markov Chain is both irreducible and positive recurrent. The proof is forthcoming and will accompany a paper to be submitted for publication.

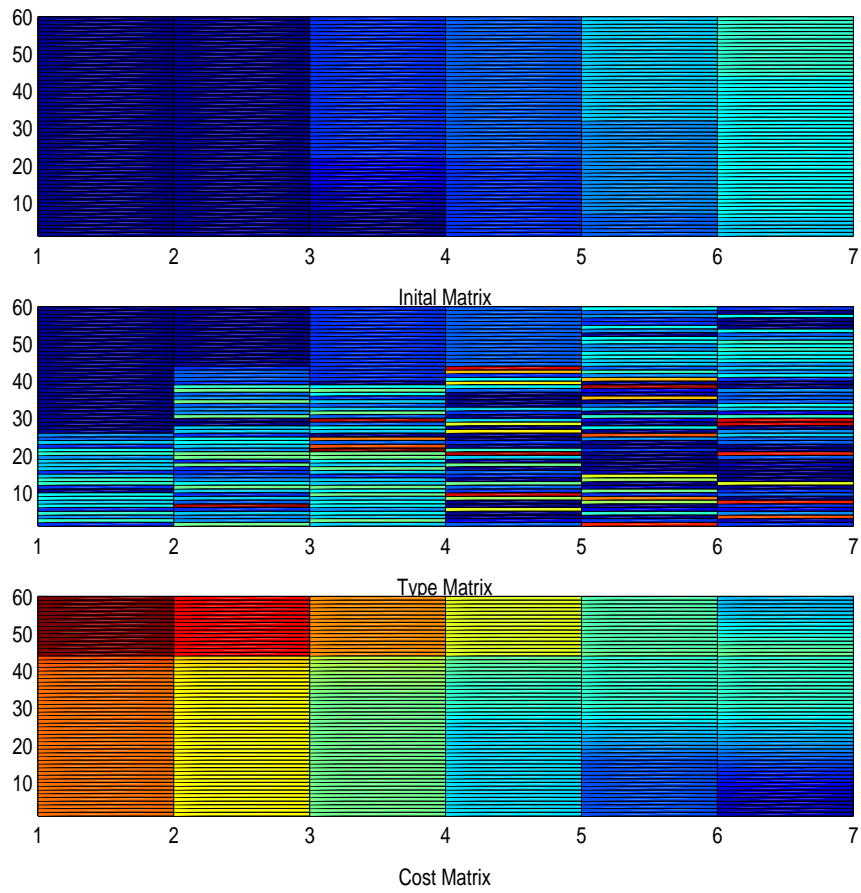
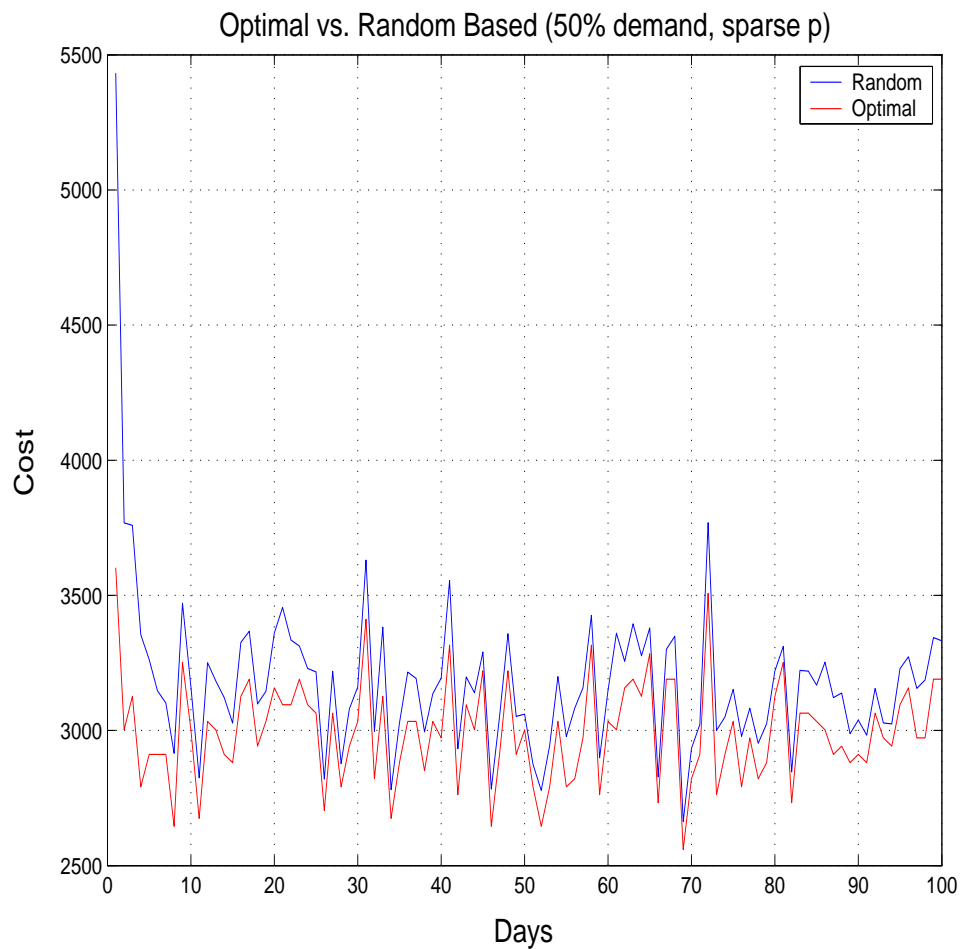


Figure 2: Random Policy: The bottom figure is a depiction of the cost matrix using in the simulations. The top figure is a color coded matrix of types of items depicting the initial state of the Type matrix. Note that highly demanded items (in dark blue) are in the most expensive locations). The center figure depicts the final state of the Type matrix.



1

Figure 3: Random Policy: This graph shows a sample path of the total daily cost (in blue). Note the sharp decline in cost as times passes by. This decline is due to an *improved* Type matrix via the policy. The red line depicts the minimum daily cost under the optimal allocation of items (i.e. under the unrealistic assumption that tickets are completely predictable).

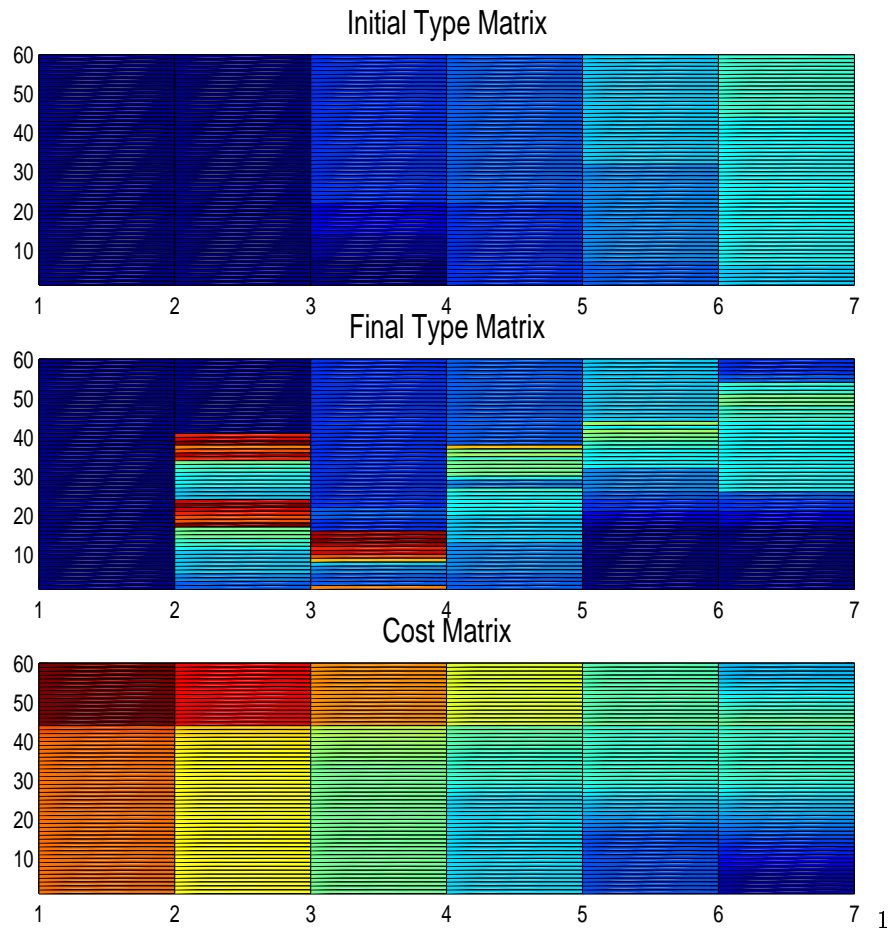
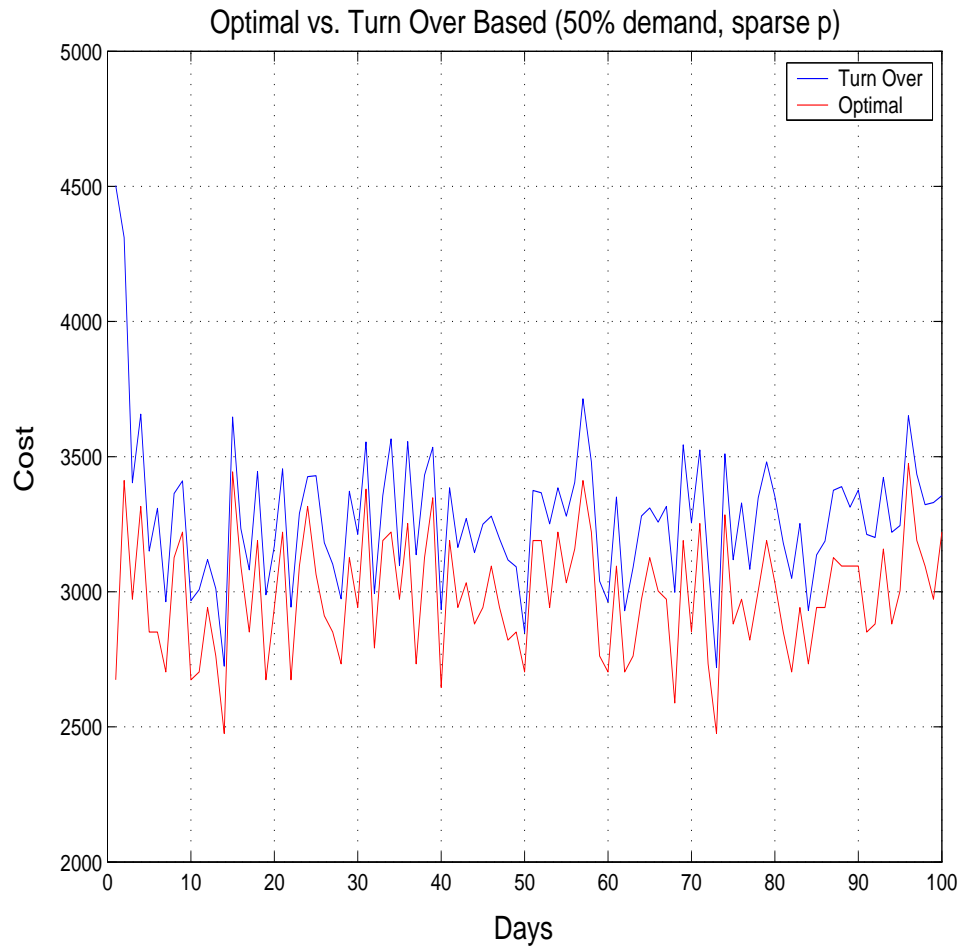
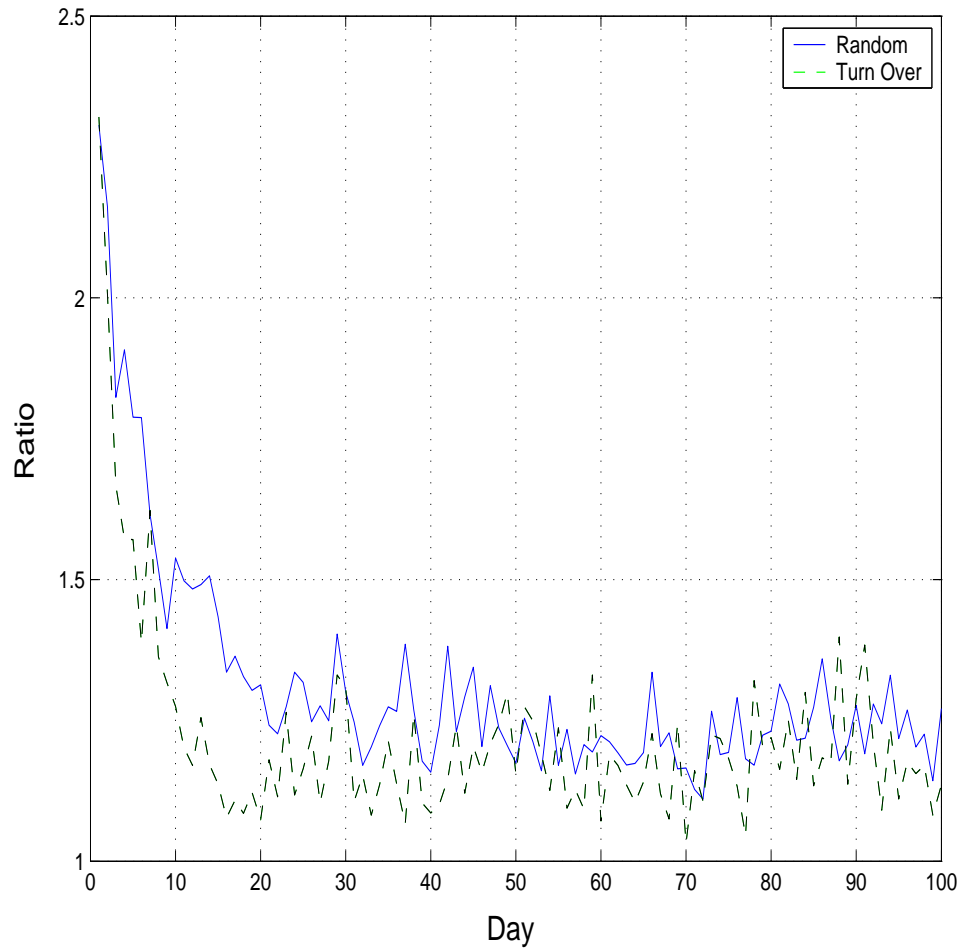


Figure 4: Turn Over Policy: The two top graphs show the initial and final Type matrices in the simulation. The cost matrix appears at the bottom.



1

Figure 5: The blue line shows the daily cost for a simulated path of tickets. The red line is the cost for the optimal placement of items (under the unrealistic assumption that tickets are completely predictable).



1

Figure 6: Random vs. Turn Over Comparison: This graph show the percentage loss in cost-wise efficiency for both the Random Policy and the Turn Over based one over the optimal policy. Each line depicts the ratio of the realized cost under the respective policy over the cost under the optimal policy.

5 Simulation Code:

5.1 Turnover Based Simulation Code:

```
clear
% Load Data (Cost Matrix)
load Cost.txt
% Size of Matrix
s = size(Cost);
N = prod(s);
% Initialize X matrix of locations occupied
% Note X(i,j) = 0 means full.
X = zeros(s);
p = [ .101 .10 .094 .09 .09 .075 .07 .05 .05 .05 .03 .03 .02 .02 .02 .02 .02
.01 .01 .01];
% Type Matrix Creator
% Frequencies
%A = 45; %Percentage of most frequent item
%p_class = [A A/2 A/3 (A/3)*.8]/100;
%p_class = [p_class 1-sum(p_class)];
%p=zeros(1,10);
%p(1:3) = p_class(1)*[.90 .07 .03];
%p(4:5) = p_class(2)*[.6 .4];
%p(6:7) = p_class(3)*[.55 .45];
%p(8:9) = p_class(4)*[.8 .2];
%p(10) = p_class(5);
% Flip X
flipX = 1-X;
TypeRand=unifrnd(0,1,s);
cp = cumsum(p);
cp = [ 0 cp];
for l=1:length(cp)-1
[ii,jj] = find(and(cp(l)<TypeRand,TypeRand<cp(l+1)));
for kk = 1: length(ii)
Type(ii(kk),jj(kk)) = 1;
end
end
```

```

Type = Type.*flipX;
% loading TypeMatrix from File
%Type = TypeMatrix;
%clear TypeMatrix;
% Items
Items = 1:20;
Class1 = 1:3;
Class2 = 4:5;
Class3 = 6:7;
Class4 = 8:9;
Class5 = 10:15;
Class6 = 15:20;
% Later we will add ClassI and ClassII
% Ticket Size
demand = .30;
T = ceil(demand*N);
% Simulation Lenght
D =1;
% Simulating Sequence of Ticekts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Saving Random seed
seed = rand('state'); % saves starting point for simulation
% sampling from multinomial
m = multirnd(T,p);
% Creates Ticket
Ticket=[];
for i=1:length(m)
Ticket =[Ticket;repmat(i,[m(i) 1])];
end
% randomizes order
permutation = randperm(T);
Ticket = Ticket(permutation);
% Create Inventory
for l=1:20
Inventory(l)=length(find(Type==l));
end
% Demand
for l=1:20

```

```

Demand(1) = length(find(Ticket==1));
end
Check = sum((Demand<Inventory))==20;
% Simulation Policy 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MissingItem=[];
DailyCost=0;
TicketList=[];
PullSequence = zeros(s);
for t=1:D
TicketList=[TicketList Ticket];
Receipt =[];
ItemsPulled = 0;
DailyCost=0;
for tt=1:T
% Find first where items are
[iv,jv]=find(Type==Ticket(tt));
if isempty(iv)
MissingItem = [MissingItem Ticket(tt)];
else
Receipt = [Receipt Ticket(tt)];
ItemsPulled = ItemsPulled+1;
CostVals = diag(Cost(iv,jv));
% Find the location with minimum cost for that item
MinCost = min(CostVals);
ij_vals = find(CostVals==MinCost);
i_val= iv(ij_vals);
j_val= jv(ij_vals);
% Updating X and Type Matrix
X(i_val(1),j_val(1)) = 1;
Type(i_val(1),j_val(1))= 0;
% Keeping Track of the Cost
DailyCost = DailyCost+MinCost;
PullSequence(i_val(1),j_val(1))=10;
PullSeq(:,:,tt) = PullSequence;
end
end
clear iv jv

```

```

EmptyX(:,:,t)=X;
TicketCost(t) = DailyCost;
% Replenishment
% We assume here that vendors randomly send replenishment items
LReceipt = length(Receipt);
Receipt = sort(Receipt);
for tt=1:LReceipt
AC = X.*Cost;
[i_zero, j_zero] = find(AC==0);
for kk=1:length(i_zero)
AC(i_zero(kk),j_zero(kk)) = max(max(AC))+1;
end
[i_val,j_val] = find(AC==min(min(AC)));
X(i_val(1),j_val(1)) = 0;
Type(i_val(1),j_val(1)) = Receipt(tt);
PullSequence(i_val(1),j_val(1)) = 0;
ReplenSeq(:,:,tt) = PullSequence;
end
m = multirnd(T,p);
% Creates Ticket
Ticket=[];
for i=1:length(m)
Ticket = [Ticket; repmat(i,[m(i) 1])];
end
% randomizes order
permutation = randperm(T);
Ticket = Ticket(permutation);
for l=1:10
Inventory(l)=length(find(Type==l));
end
% Demand
for l=1:10
Demand(l) = length(find(Ticket==l));
end
Check = sum((Demand<Inventory))==10;
TurnOver(:,:,t)=Type;
end

```

5.2 Current Policy Simulatin Code

```
clear
% Load Data (Cost Matrix)
load Cost.txt
% Size of Matrix
s = size(Cost);
N = prod(s);
% Initialize X matrix of locations occupied
% Note X(i,j) = 0 means full.
X =zeros(s);
p = [ .101 .10 .094 .09 .09 .075 .07 .05 .05 .05 .03 .03 .02 .02 .02 .02 .02
.01 .01 .01]
% Type Matrix Creator
% Frequencies
%A = 45; %Percentage of most frequent item
%p_class = [A A/2 A/3 (A/3)*.8]/100;
%p_class = [p_class 1-sum(p_class)];
%p=zeros(1,10);
%p(1:3) = p_class(1)*[.90 .07 .03];
%p(4:5) = p_class(2)*[.6 .4];
%p(6:7) = p_class(3)*[.55 .45];
%p(8:9) = p_class(4)*[.8 .2];
%p(10) = p_class(5);
% Flip X
flipX = 1-X;
TypeRand=unifrnd(0,1,s);
cp = cumsum(p);
cp = [ 0 cp];
for l=1:length(cp)-1
[ii,jj] = find(and(cp(l)<TypeRand,TypeRand<cp(l+1)));
for kk = 1: length(ii)
Type(ii(kk),jj(kk)) = 1;
end
end
Type = Type.*flipX;
% loading TypeMatrix from File
%Type = TypeMatrix;
```

```

%clear TypeMatrix;
% Items
Items = 1:20;
Class1 = 1:3;
Class2 = 4:5;
Class3 = 6:7;
Class4 = 8:9;
Class5 = 10:15;
Class6 = 15:20;
% Later we will add ClassI and ClassII
% Ticket Size
demand = .30;
T = ceil(demand*N);
% Simulation Lenght
D =5;
% Simulating Sequence of Ticekts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Saving Random seed
seed = rand('state'); % saves starting point for simulation
% sampling from multinomial
m = multirnd(T,p);
% Creates Ticket
Ticket=[];
for i=1:length(m)
Ticket =[Ticket;repmat(i,[m(i) 1])];
end
% randomizes order
permutation = randperm(T);
Ticket = Ticket(permutation);
% Create Inventory
for l=1:20
Inventory(l)=length(find(Type==l));
end
% Demand
for l=1:20
Demand(l) = length(find(Ticket==l));
end
Check = sum((Demand<Inventory))==20;

```

```

% Simulation Policy 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MissingItem=[];
DailyCost=0;
TicketList=[];
for t=1:D
TicketList=[TicketList Ticket];
Receipt=[];
ItemsPulled = 0;
DailyCost=0;
for tt=1:T
% Find first where items are
[iv,jv]=find(Type==Ticket(tt));
if isempty(iv)
MissingItem = [MissingItem Ticket(tt)];
else
Receipt = [Receipt Ticket(tt)];
ItemsPulled = ItemsPulled+1;
CostVals = diag(Cost(iv,jv));
% Find the location with minimum cost for that item
MinCost = min(CostVals);
ij_vals = find(CostVals==MinCost);
i_val= iv(ij_vals);
j_val= jv(ij_vals);
% Updating X and Type Matrix
X(i_val(1),j_val(1)) = 1;
Type(i_val(1),j_val(1))= 0;
% Keeping Track of the Cost
DailyCost = DailyCost+MinCost;
end
end
clear iv jv
EmptyX(:,t)=X;
TicketCost(t) = DailyCost;
% Replenishment
% We assume here that vendors randomly send replenishment items
LReceipt = length(Receipt);
randind = randperm(LReceipt);

```

```

Replenish = Receipt(randind);
for tt=1:LReceipt
AC = X.*Cost;
[i_zero, j_zero] = find(AC==0);
for kk=1:length(i_zero)
AC(i_zero(kk),j_zero(kk)) = max(max(AC))+1;
end
[i_val,j_val] = find(AC==min(min(AC)));
X(i_val(1),j_val(1)) = 0;
Type(i_val(1),j_val(1)) = Replenish(tt);
end
m = multirnd(T,p);
% Creates Ticket
Ticket=[];
for i=1:length(m)
Ticket = [Ticket; repmat(i,[m(i) 1])];
end
% randomizes order
permutation = randperm(T);
Ticket = Ticket(permutation);
for l=1:10
Inventory(l)=length(find(Type==l));
end
% Demand
for l=1:10
Demand(l) = length(find(Ticket==l));
end
Check = sum((Demand<Inventory))==10;
TurnOver(:,t)=Type;
end

```

References

- [1] Daniels R, Rummel J, Schartz R. *A Model for Warehouse Order Picking*. European Journal of Operational Research, 105,1998, p 1-17.
- [2] Hausman W, Shwartz R, Graves S. *Optimal Storage Assignment in Automated Warehousing Systems*, Management Science, 22, 1976, p 629-638.
- [3] Kaitash J. *Management Policies and Warehouse Requirements: A Case Study*. Journal of Purchasing and Materials Management, Spring 1990,p 27-31.
- [4] Kellere H. *Bottleneck Quadratic Assignment Problems and the Bandwidth Problem*. Asia Pacific Journal of Operational Research, 15, 1998, p 165-177.
- [5] Resnick, Sidney I., *Adventures in Stochastic Processes*, 1994, Birkh auser, Boston.
- [6] Thonemann U, Brandeau M. *Optimal Storage Assignment Policies for Automated Storage and Retrieval Systems with Stochastic Demands*, Management Science, Jan 1998, Vol 44 no. 1, p 142-148.
- [7] Van Der Berg J, Gademann A. *Simulation Study of Automated Storage/Retrieval Sytems*, Int. Journal of Production Resources, 2000, Vol 38 no 6, p 1339-1356.
- [8] Winston W, *Operations Research: Applications and Algorithms*,1987,Duxbury Press, Boston p 110-167.