Allocating Manufacturing Capacity by Solving a Dynamic Stochastic Multi-Knapsack Problem

Thomas C. Perry
Joseph C. Hartman
Lehigh University

Report No. 04T-009

Allocating Manufacturing Capacity by Solving a Dynamic Stochastic Multi-Knapsack Problem

Thomas C. Perry and Joseph C. Hartman
Industrial & Systems Engineering, Lehigh University, Bethlehem, Pennsylvania

Revenue management problems have been extensively studied and applied to the airline, hotel and rental industries. These problems generally maximize system profitability by accepting or rejecting arrivals based on their reward and how many resources they utilize. If a deadline is reached and a resource goes under utilized, potential revenue is lost. Problems in the literature generally involve the selection of arrivals that would exist in the system up to some specific deadline (could be finite or infinite). This is often practical for the case of the transportation industry (airline, cargo transport, and others), but is often not applicable in continuous manufacturing situations. Typically, an arrival (production order) will enter a manufacturing facility and stay for some period of time, utilizing resources (production lines). To account for this attribute, one can view a manufacturing facility's capacity as a series of knapsacks with orders modeled as arrivals that can be placed in these knapsacks. Accept and reject decisions are made based on the utilization of the capacity window, or a knapsack representing system capacity over some discrete period of time. Since the capacity window may consist of past arrivals, accept and reject decisions are considered beyond the current period. As future arrivals are uncertain, a stochastic dynamic program is used to model this situation. We present several strategies employed to solve the model and insight gained from the solution of various situations encountered in the semiconductor manufacturing sector.

1.0 Background

This investigation is motivated by a problem in the semiconductor industry where a wafer fabrication line can produce multiple products. State of the art semiconductor fabrication lines can cost over $3B, resulting in very high fixed costs of operation. This necessitates that capacity be fully utilized and consist of the right mix of products. This is a difficult task due to the long lead-time between design and production and the lengthy cycle-time of the product. With long cycle times, selecting the right product to most effectively utilize the available resource is critical. The literature has closely studied the issue of utilization, with a number of papers dealing with production scheduling, but considering the mix or portfolio of products has been largely overlooked. In this paper, we consider the problem of having production orders arrive over time, defined by capacity needs and expected revenues. The orders must be accepted, upon which they use capacity and generate revenue, or rejected, freeing up capacity for later orders.

This can be generalized to any manufacturing setting that uses a common set of resources to manufacture multiple products with varying cycle times. It is important to determine the best policy of accepting or rejecting new sources of revenue. This policy can be studied under uncertain demand in a constantly changing system. The dynamics of the system would include several factors including the level of capacity utilization, the volume, the profit and the risk associated with the arrival.

In the classic knapsack problem, items of known size and reward are placed in a knapsack of known capacity. This static and deterministic problem is well studied (Martello and Toth [1]). Kleywegt and Papastravrou [2] [3] extend this concept to the case where items arrive over time with unknown size and rewards. This type of problem is classified as the dynamic stochastic

knapsack problem (DSKP). Several classes of problems are associated with the DSKP and have been summarized by Weatherford and Bodily [4] as perishable asset revenue management (PARM) problems. All of these problems however, deal with a single knapsack with a certain deadline. For example, consider a transportation problem to haul packages from one location to another. Items arrive and are either accepted or rejected for an upcoming shipment. Once an item is accepted, it waits for the shipment to take place. The time of the shipment then becomes the deadline.

Using the knapsack analogy, a continuous manufacturing line can be viewed as a set of multiple knapsacks. These knapsacks represent the capacity of the manufacturing line over consecutive periods of time, such as months or quarters. This model is required as there are generally no deadlines in which the capacity departs, such as a truck or an airplane. Rather, orders depart the system once their production run is completed.

Items, representing orders, arrive and if accepted, are placed in a knapsack. If an order spans multiple periods, then it may fill multiple knapsacks in time until it departs, representing the end of a production run. During its production, more orders may arrive and if capacity is available, may be accepted or rejected. If an item arrives but there is no capacity, it is rejected. Hence, the acceptance of an order must consider the future arrival of orders, as the product mix greatly affects profitability.

Ross and Tsang [5] study a problem that is analogous to this problem. They study the effective use of bandwidth through a communication switching network. Multiple classes of calls can arrive to which bandwidth is allocated. The length of the call is random. Ross and Tsang approach this problem as a Markov Decision Process (MDP) and determine a threshold policy for 2 given classes of calls. As accepted orders to the manufacturing systems are generally contracted, and not random, we consider a different approach by incorporating what we term a capacity window. The capacity window is a representation of capacity utilization over multiple time periods. The capacity is bounded in each period, resulting in the use of multiple knapsack constraints to model capacity. This allows us to study manufacturing capacity in this window in a rolling-horizon fashion.

To illustrate the need for multi-period knapsacks, consider a two period problem. Two classes of objects are known to arrive each period. Once an object is accepted, it will remain in the system for two periods (manufacturing cycle time). The initial knapsack (manufacturing capacity) is empty at the beginning of the first period. Given,

t	time period
$i \in \{1,2\}$	arrival class i
$X_{i,t} \in \{0,1\}$	accept/reject of arrival class t in time period t
$R_i \in \{300,200\}$	reward of class i
$C_i \in \{0.75,0.5\}$	capacity needs of class i
$B = 1$	maximum capacity

This becomes a two period static multiple knapsack problem with the objective to maximize the reward, or

$$\max Z = \left(X_{1,1} + X_{1,2}\right)R_1 + \left(X_{2,1} + X_{2,2}\right)R_2$$

and is subject to the following multi-knapsack capacity constraints

$$X_{1,1}C_1 + X_{2,1}C_2 \le B$$

$$\left(X_{1,1} + X_{1,2}\right)C_1 + \left(X_{2,1} + X_{2,2}\right)C_2 \le B$$

Figure 1 illustrates the possible choices in the first two periods. Given an empty system at the beginning of the first period, the optimal decision would be to accept object one. However, if time period two is considered, you can see that the optimal decision is to accept object two in the first period and then accept another object two in the second period. Or, $X_{1,1} = 0$, $X_{2,1} = 1$, $X_{1,2} = 0$ and $X_{2,2} = 1$. This illustrates the need to look at multiple periods, which is even more critical under the assumption of dynamic arrivals.

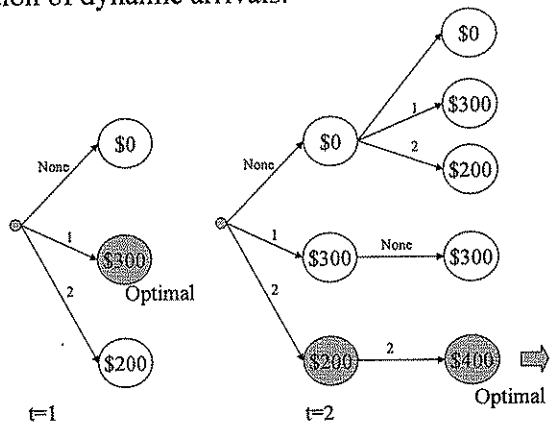

Figure 1 – One and two period representation of order acceptance problem.

The incorporation of the capacity window allows the extension of the DSKP to a dynamic stochastic multiple knapsack problem (DSMKP). In this paper, we formulate the capacity reservation problem as a DSMKP. As we are motivated by real problems in industry, we explore ways in which to efficiently solve the associated dynamic program, which has an exponential run time in the number of arrivals per period.

Section **Error! Reference source not found.** will give an overview of the model. Section 3.0 will discuss the limitations of the model. The use of a capacity window allows the use of several techniques that can be used to simplify the implementation of the model. These techniques will be described in section 4.0. Finally, the results of the model will be presented in section 5.0.

2.0 A Dynamic Stochastic Multiple Knapsack Problem

We use discrete time intervals that coincide with the arrivals of new orders (i.e. monthly or quarterly). Under this assumption, the state of the system is defined by the utilization of capacity at each period of time. Note that this is equivalent to modeling the available capacity of the system.

2.1 The System State and Capacity Window

The "capacity window" noted earlier, refers to the state of the system, as it is the number of periods of capacity tracked through time. This can be represented as a capacity utilization vector (also referred to as the state vector or the state of the system):

$$\mathbf{S} = [b_1, b_2, b_3, b_4],$$

where b_t is the utilized capacity in period t. Decisions on orders occur at the end of each period, the decision is to either accept or reject the order. Accepting an order reserves capacity, thereby increasing utilization of the system, for the periods defined by the order's contract. Thus, if an order is accepted which consumes w units of capacity for two periods beginning in period 2, the resulting state of the system is:

$$\mathbf{S} = [b_2 + w, b_3 + w, b_4, b_5].$$

As with any perishable asset, the capacity is either consumed or left underutilized as the system transitions in time. Our capacity window allows for the system to be studied over time, such that the utilization of the system merely shifts in time. For example, given our original state of the system, if no orders are accepted, then at the next decision epoch, the state of the system is:

$$\mathbf{S} = [b_2, b_3, b_4, b_5],$$

which represents a translation of the capacity window by one time period. An illustration of this capacity window can be found in Figure 2. In this illustration, the four period capacity window represents the utilized capacity of the system over four time periods. At the end of each time period, the capacity merely shifts one time period.

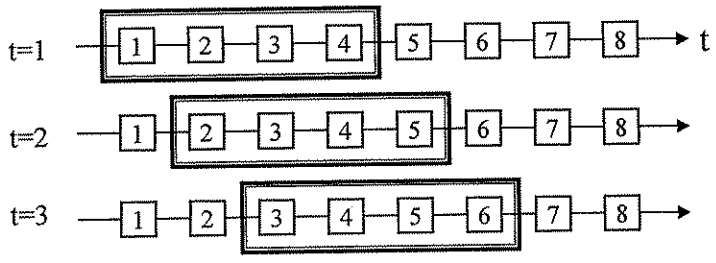

Figure 2 – The system capacity utilization is captured in a 4-period Capacity Window.

There is a maximum available capacity for each time period that cannot be exceeded, representing the maximum available resources of the system. The maximum capacity at each time period is represented by a knapsack constraint. Thus, the capacity of the system is represented by a series of K knapsacks. If b_k is the capacity of knapsack k, the capacity of the system can be written as:

$$\mathbf{S} = [b_1, b_2, \ldots, b_K].$$

If an order is to be accepted, the capacity that is needed to fulfill the order must satisfy the knapsack constraint for all time periods within the capacity window.

2.2 State Transition

This process is illustrated in more detail in Figure 3. The knapsacks in the capacity window are labeled 1 through K, where K is the number of knapsacks (or, correspondingly the number of time periods in which the capacity utilization of the system is being tracked). A transition in time is illustrated in Figure 3, where the capacity in knapsack $k+1$ of the pervious time period becomes the initial capacity of knapsack k in the current time period. This represents orders that are still in the system that have not completed. In addition to the translation of the capacity window, the capacity of orders that are accepted in the previous time period are added to the knapsacks based on the orders requirement for the system capacity. This is represented as small boxes in Figure 3. The knapsack constraint must be met in each time period or the order must be rejected.

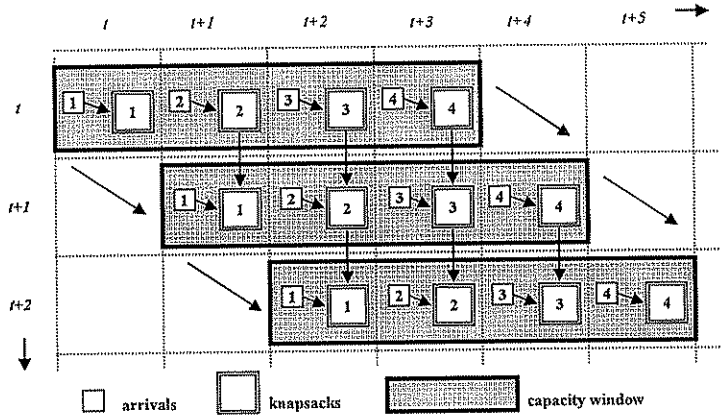

Figure 3 – Snapshot of utilized capacity over time defines capacity window.

At each time period, several decisions may exist to either accept or reject order arrivals. The accept-reject combinations are dependent on the number of order arrivals per period, and are defined as decision set, δ. For example, two arrivals would have four in the decision set: (1) accept none (empty set), (2) accept arrival one and reject arrival two, (3) accept arrival two and reject arrival one and (4) accept both arrivals. In general, the number of decision sets is 2^N, where N is the number of arrivals. This model considers that there is some probability that an order may arrive in a given period. The probabilistic arrival/no arrival combinations are defined

as the arrival set ψ. The number of objects in the arrival set is equal in number to the decision set, comprising all combinations of orders arriving or not arriving (resulting in a total of 2^N possible arrival combinations). The transition paths from state $\mathbf{S}$ to $\mathbf{S}'_{\delta,\psi}$ is illustrated in Figure 4. $\mathbf{S}'_{\delta,\psi}$ is the next state, or the capacity utilization of the system, resulting from a translation of the capacity window and the capacity from orders resulting from the intersection of the decision set δ and the arrival set ψ.

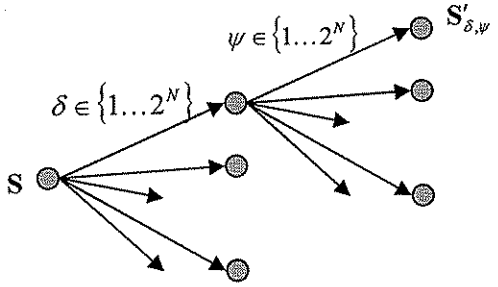

Figure 4 – State transitions consist of a decision set and an arrival set which is used to determine the expected reward given the probability of the arrivals.

2.3 Expected Reward

Associated with each arrival set ψ is a probability P_ψ. This probability is the product of the probability of arrival, p_n, or the probability of not arriving, $1-p_n$, for all the orders in the arrival set. P_ψ can be calculated as:

$$P_\psi = \prod_{n \in \psi} (p_n) \prod_{n \notin \psi} (1 - p_n).$$

The transition from $\mathbf{S}$ to $\mathbf{S}'_{\delta,\psi}$ as shown in Figure 4 results in a potential reward. In order to receive a reward, two things must occur: (1) an order must arrive, which is defined by the arrival set ψ and (2) we must accept the order, which is defined by our decision set δ. The reward is the combination of arrivals that are both accepted and arrive, or:

$$R_{\delta,\psi} = \sum_{n \in \delta \cap \psi} r_n .$$

2.4 Dynamic Program Recursion

Let $V_t(\mathbf{S})$ represent the maximum expected discounted profit through time horizon T, assuming one starts with a system utilized for K periods at levels b_1, b_2, through b_K at time t. At each stage, time period, we maximize the reward over the decision set δ, which results in a new state $\mathbf{S}'_{\delta,\psi}$, depending on the arrival set ψ, as:

$$V_t(\mathbf{S}) = \max_{\delta \left| \{ \mathbf{S}'_{\delta,\psi} \in B \} \right.} \left[\sum_{\psi} P_{\psi} \left(R_{\delta,\psi} + V_{t+1}(\mathbf{S}'_{\delta,\psi}) \right) \right],$$

where $\mathbf{S}'_{\delta,\psi}$ must satisfy the knapsack constraint in each time period (must be an element in the set of valid states B). We discuss this now.

2.5 Capacity Constraints

All accepted arrivals in δ are subject to the system capacity window constraint, represented in our recursion as:

$$\mathbf{S}'_{\delta,\psi} \in B,$$

where $\mathbf{S}'_{\delta,\psi}$ is the resulting capacity window based on the decision set δ and the arrival set ψ. B is the set of valid system states (no violation of capacity)

As illustrated in Figure 3, the capacity window is bounded by the capacity of K knapsacks. As orders arrive to the system, the decision to accept or reject is made based on the future knapsack constraints. If there is insufficient capacity the item is rejected. If there is sufficient capacity in the system for the multi-period capacity of the order arrival, the order is evaluated for possible inclusion.

Let b_k be the utilization of knapsack k. The initial utilization of the knapsack is the sum of the capacity of orders that are still in the system in the $k+1$ knapsack of the previous time interval, $t-1$ (refer to Figure 3). The required capacity of any order arrival that is accepted in the previous period is added to this and compared to the capacity limit.

Let the capacity of arrival n can be expressed as a vector:

$$\mathbf{C}_n = \left[w_{n,1}, w_{n,2}, \ldots, w_{n,K} \right],$$

where $w_{n,1}$, $w_{n,2}$, through $w_{n,K}$ is the capacity for each time period t in the capacity window. As mentioned previously, the decision to accept an order occurs at the end of the current period. However, the order capacity is not added to the system state until the next time period. To keep the multi-knapsack capacity balance, the current state vector needs to be translated on period in time. The current state vector is given by:

$$\mathbf{S} = \left[b_1(t), b_2(t), \ldots, b_K(t) \right].$$

This state vector is translated one period in time, $\hat{\mathbf{S}}(t)$, as shown below:

$$\hat{\mathbf{S}} = [b_2(t-1), b_3(t-1), \ldots, b_K(t-1), 0].$$

The last knapsack is zero because this is new capacity that becomes available which is beyond the capacity window. Once the current state capacity is translated, the arrival capacity utilization vector:

$$\sum_{n \in \delta \cap \psi} \mathbf{C}_n = \sum_{n \in \delta \cap \psi} [w_{n,1}(t), w_{n,2}(t), \ldots, w_{n,K}(t)],$$

can be added and the resulting state vector is:

$$\mathbf{S}'_{\delta,\psi} = \hat{\mathbf{S}} + \sum_{n \in \delta \cap \psi} \mathbf{C}_n .$$

Referring to Figure 3, this constraint for each individual knapsack can be expressed as:

$$b_k(t) = \left\{ \begin{array}{ll} b_{k+1}(t-1) + \sum_{n \in \delta \cap \psi} w_{n,k}(t). & \text{for all } k \in \{1,2,\ldots,K-1\} \\ \sum_{n \in \delta \cap \psi} w_{n,k}(t). & \text{for } k = K \end{array} \right\} \leq \bar{b}_k .$$

This can be implemented in a stochastic dynamic program recursion that can be solved backwards. The key to implementing this stochastic dynamic program is the ability to keep track of the state $\mathbf{S}$ of the capacity window.

3.0 Model Limitations

This model poses two difficulties, namely the enormous number of calculations and infinite state space. This is due in part to the general flexibility of the model to allow a varying number of arrivals per time period. It is also due in part to the implementation of the capacity window in which multiple knapsack constraints need to be considered.

The number of calculations for each state and for each time period can be derived from all possible decision sets at each state node. The number of decisions is calculated as 2^N. The stochastic nature of this problem adds another factor of complexity to determine the expected revenue at each state node. The dynamic program algorithm must iterate over all possible arrival sets for each decision. Therefore, the total calculations per state per time period, is 2^{2N}. If the model runs over T periods, the resulting calculations would be $2^{2N}T$. Therefore, the number of calculations grows exponentially with the number of arrivals. This is made worse by the maximum number of states in a given time period that is dependent on how we define capacity, which we discuss later.

It is a known issue that the state space of dynamic programs can quickly become unmanageable, commonly referred to as the curse of dimensionality. This is especially true in this model with the capacity window. Let us assume a one period (one knapsack) problem where the arrival

number is fixed for each time period and all have equal capacity. A node in the state diagram could be represented in terms of the number of items in the knapsack resulting in a finite number of states. Representing the state by the amount of utilized capacity leads to an infinite number of states since capacity utilization is continuous on [0,1]. Multiple knapsack constraints further complicate the state space.

The infinite states could be reduced to a finite set utilizing aggregation techniques like the ones proposed by Bean et al. [6] or Kim and Smith [7]. If the capacity utilization is divided into a discrete number of capacity buckets, the state space could be described and the number of states becomes finite, although very large. The approach using the discrete buckets of capacity will be discussed in the next section in greater detail.

But, to illustrate the complexity, let β represent the number of discrete capacity buckets. The total number of states would then be β^K, where K is the number of knapsacks constraints needed to describe the system. Combining this with the number of calculations shown earlier, the total calculations would be $2^{2N}T\beta^K$.

The magnitude of calculations per period for an example with $\beta = 20$, $K = 8$, and $N = 5$, would be:

$$Y = 2^{10}20^8 \cong 26x10^{12}.$$

The dynamic program would need to run over a number of periods at least equal to the number of knapsacks. Let us assume that one calculation by a computer takes 1nS (1e-9 sec). This many calculations would take approximately 7.5 hours per period to calculate. This is certainly large and would not satisfy the requirements of a real time calculation. (Note that while we may not require real time decisions, the user presumably will have to make accept or reject decisions within a reasonable time period (1-2 business days).)

4.0 Implementation

Several techniques have been used to simplify the implementation of the model. These techniques simplify the calculations and the looping structure of the dynamic program.

4.1 Capacity Buckets

As previously noted, the percent of capacity utilization is continuous on $[0,1]$, taking on an infinite number of possible states. This complicates the implementation of a dynamic program. Therefore, capacity utilization is defined by dividing the capacity into discrete buckets β. This aggregates the state space into a large finite number of nodes. An example of this aggregation can be found in Table 1 where the capacity is broken into four buckets, $\beta = 4$.

b_k **capacity utilization of knapsack** k	**Bucket** ($\beta = 4$)

$b_k = 0$	0
$0 < b_k \leq 0.33$	1
$0.33 < b_k \leq 0.67$	2
$0.67 < b_k \leq 1.00$	3

Table 1 - Capacity buckets aggregate infinite capacity utilization into discrete capacity buckets.

The amount of capacity utilization that the capacity buckets represents, determines the complexity of the dynamic program. The smaller the capacity bucket size (finer granularity) the larger the number of states. Problems that would be solved for larger manufacturing situations would have 20 or more buckets, which would translate to a bucket size of 5% capacity utilization. Order arrivals typically would utilize up to 25% capacity, with a typical capacity utilization around 2-5%. The bucket size needs to be small enough to be able to represent low volume orders.

4.2 The System Capacity Window

The system state vector continues with the same notation; however, b_k now represents the capacity bucket, or the aggregate capacity utilization level the system:

$$\mathbf{S} = [b_1, b_2, \ldots, b_K].$$

For example, Figure 5 illustrates the state $\mathbf{S} = [3,2,3,0,1]$ with $K = 5$ knapsacks and $\beta = 4$ capacity buckets.

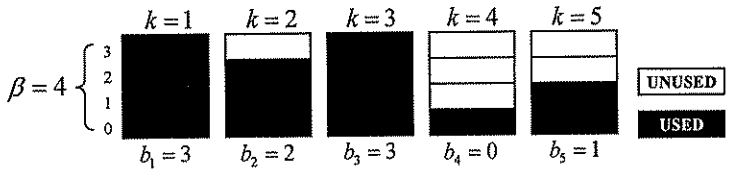

Figure 5 – Representation of state vector $\mathbf{S} = [3,2,3,0,1]$ with $K = 5$ knapsacks with $\beta = 4$ capacity buckets.

This state vector can be represented by a single numerical value by concatenating together the amount of capacity buckets that are used for each time period of the capacity window. The numerical representation of the state vector for the example shown in Figure 5 is 32301.

4.3 The Arrival Capacity Window

This concept can also be applied to the capacity window $\mathbf{C}_n$ of arrival n. The arrival capacity vector notation stays the same; however, $w_{n,k}$ now represents the capacity bucket, or the aggregate capacity utilization level required by the order arrival:

$$\mathbf{C}_n = [w_{n,1}, w_{n,2}, \ldots, w_{n,k}].$$

Figure 6 illustrates the capacity utilization of the arrival capacity window $\mathbf{C}_n = [1,0,1,0,0]$ with $K = 5$ knapsacks and $\beta = 4$ capacity buckets.

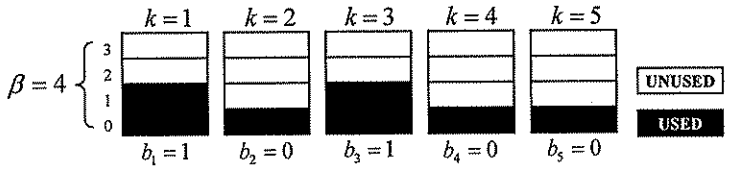

Figure 6 - Representation of arrival capacity utilization vector $\mathbf{C}_n = [1,0,1,0,0]$ with $K = 5$ knapsacks with $\beta = 4$ capacity buckets.

As with the state vector, the arrival capacity vector can be represented by a single numerical value by concatenating together the amount of used capacity buckets. The numerical representation of the state vector for the example shown in Figure 6 is 10100.

4.4 State Transition

When considering an arrival, the decision is made in the current period and the arrival begins to utilize capacity any of the ensuing periods. In order to add the arrival capacity window to the system capacity window, the system capacity window needs to be translated one time period. As explained earlier, this represents orders that are still in the system which have not been completed. The translation of the capacity word described in Figure 5 is shown in Figure 7.

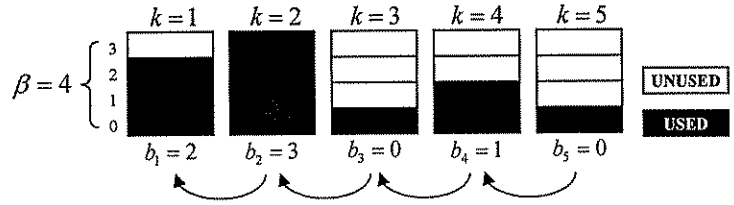

Figure 7 – Translation of the state vector $\mathbf{S} = [3,2,3,0,1]$ by one time period to $[2,3,0,1,0]$

Therefore, the state numerical representation of 32301 when translated one period would be 23010.

4.5 State Transition with Arrivals

With both the system and arrival capacity utilization represented by a single number, the resulting state can be determined by simple addition. This is illustrated in Figure 8.

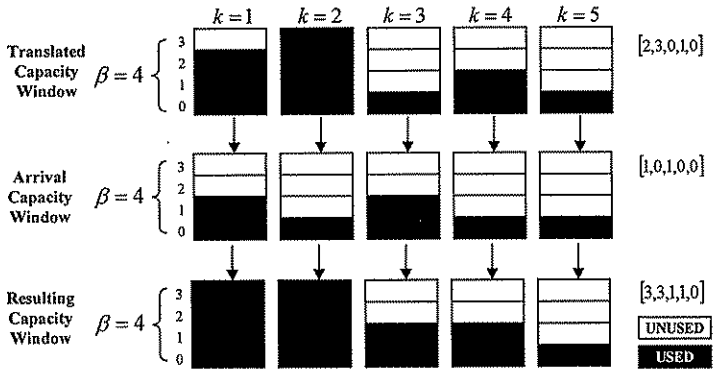

Figure 8 - Example of the system state transition with an accepted arrival.

The translated state vector represented by 23010 and the arrival capacity vector represented by 10100, are added to obtain the resulting state of 33110.

4.6 Numerical Representation

It is noted that we implemented the dynamic program using numerical representations of the state vector. The benefits are in flexibility of the model. See Perry and Hartman [8] for more details.

4.7 Reduction of State Nodes in the Dynamic Program

To accommodate the large number of states that are possible with this flexible DSMKP, another technique is introduced. Even though there are potentially a large number of states that can theoretically exist in the state space, in reality, only a portion of them are actually used (given the number of periods). In order to limit the number of states that need to be evaluated in the stochastic dynamic program, a simple routine is run to build the network forward to capture the valid state nodes. With the establishment of valid state nodes, the stochastic dynamic program can operate backwards only on valid network nodes.

Of course, this does require the construction of the network, which can be done by assuming all future orders will arrive (deterministic case). Beginning with the initial state, the next time period states can be determined by evaluating all the possible decision combinations. Continuing in this fashion the network can be built for subsequent time periods. In doing this, the early time periods will have a relatively small number of states, growing to a large number of states as time progresses. To determine the minimum number of time periods in which the valid states would approach the number of possible states could be determined by solving for T in

$$N^T = \beta^K,$$

where N^T is the maximum number of valid states at time period T. This will give the lower bound on the number of time periods, since in a real situation, several decision combinations will most likely results in redundant states.

The time to build the network is depends on the number of decision sets as shown in an Figure 9 and Table 2.

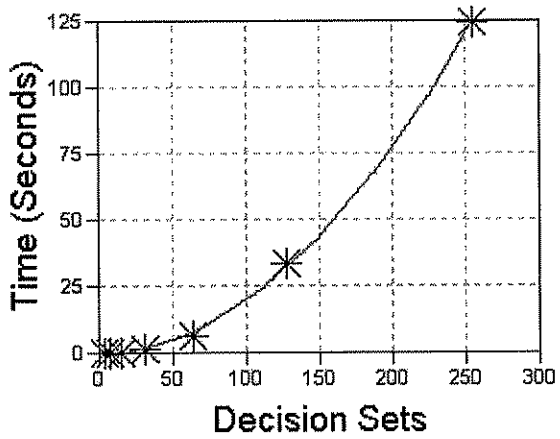

Figure 9 - Build Network Time dependence on decision sets

T	N	β	K	β^K	δ	Build Network Time (sec)
6	2	11	3	7986	4	0.09
6	3	11	3	7986	8	0.17
6	4	11	3	7986	16	0.45
6	5	11	3	7986	32	1.67
6	6	11	3	7986	64	6.74
6	7	11	3	7986	128	33.69
6	8	11	3	7986	256	124.39

Table 2 - Time to build a valid network

5.0 Results

As shown in section 3.0, the total calculations is given by

$$2^{2N} T\beta^K .$$

The complexity of the dynamic program then depends on four factors: the time horizon, the number of arrivals, the number of capacity buckets and the number of knapsacks. The model was run varying these four parameters and the results are shown in Table 3. For each model run, the execution time was recorded. The DP was also run in two different ways: DP type full and DP type valid. In the DP type full, all possible states are evaluated at each time period. In the

DP type valid, the state network is reduced as described in section 4.7. A fixed number of potential arrivals were assumed each period. Figure 10 summarizes the dependence of the different parameters on the execution time.

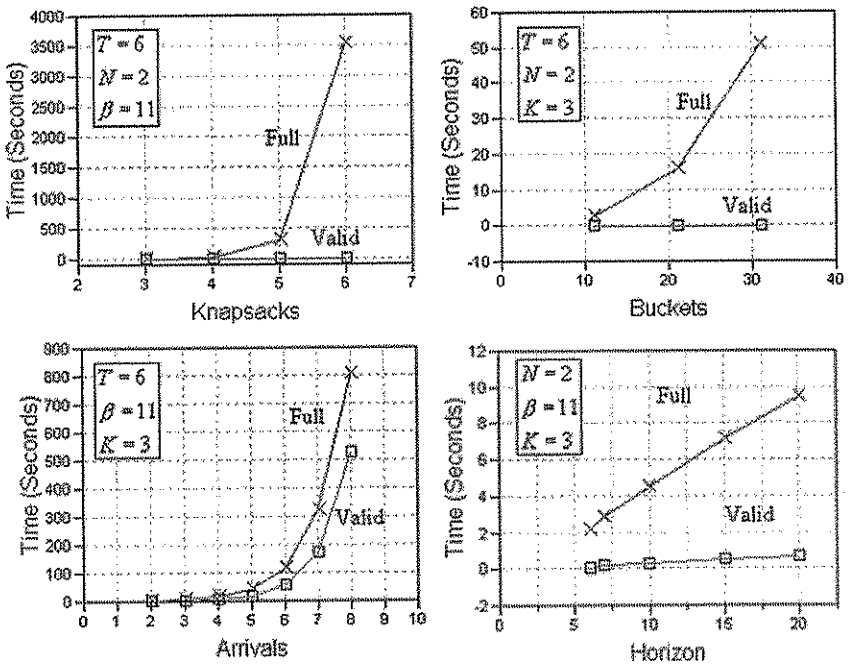

Figure 10 - Execution time for different number of knapsacks, buckets, arrivals and horizons.

From these results it becomes evident that two factors influence the execution time. The first is the number of states and the second is the number of arrivals. Increasing either the number of knapsacks (states) or the number of arrivals substantially increases the execution time. Increasing the number of buckets also has an affect, but to a lesser extent. In general, the execution time can be plotted as a function of the number of states and arrivals as shown in Figure 11.

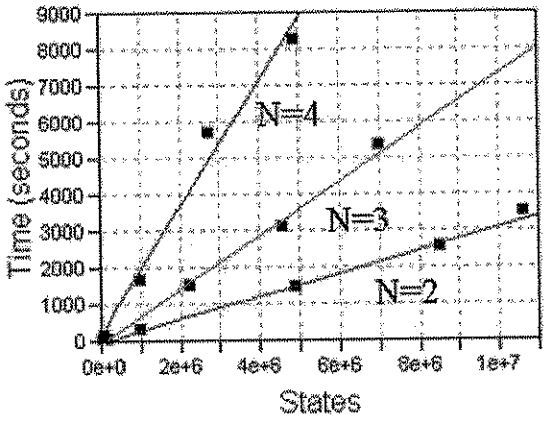

Figure 11 - Execution time dependence on the number of states and number of arrivals

The reduction in execution time for the DP type valid versus the DP type full is a direct correlation to the number of states for each DP. Reducing the number of states clearly results in a reduction of the execution time. There is an advantage to determining the valid states prior to performing the DP. This figure also illustrates that increasing the number of arrivals, the execution time quickly grows exceptionally large.

T	N	β	K	δ	ψ	$\delta\psi$	Full DP States	Valid DP States	Full DP Time (sec)	Valid DP Time (sec)
6	2	11	3	4	4	16	7,986	81	2.19	0.08
6	2	11	4	4	4	16	87,846	81	24.06	0.08
6	2	11	5	4	4	16	966,306	81	287.50	0.06
6	2	11	6	4	4	16	10,629,366	81	3453.53	0.08
6	2	11	3	4	4	16	7,986	81	2.19	0.08
6	3	11	3	8	8	64	7,986	151	5.76	0.45
6	4	11	3	16	16	256	7,986	437	16.39	8.45
6	5	11	3	32	32	1,024	7,986	345	31.56	18.34
6	6	11	3	64	64	4,096	7,986	450	107.45	57.83
6	7	11	3	128	128	16,384	7,986	544	376.44	243.17
6	8	11	3	256	256	65,536	7,986	364	960.45	337.52
6	2	11	3	4	4	16	7,986	81	2.19	0.08
7	2	11	3	4	4	16	9,317	104	2.86	0.11
10	2	11	3	4	4	16	13,310	168	4.49	0.20
15	2	11	3	4	4	16	19,965	273	7.14	0.31
20	2	11	3	4	4	16	26,620	442	9.51	0.51
6	2	11	3	4	4	16	7,986	81	2.19	0.08
6	2	21	3	4	4	16	55,566	110	15.84	0.11
6	2	31	3	4	4	16	178,746	114	49.09	0.11
5	2	11	3	4	4	16	6,655		1.78	
6	2	11	3	4	4	16	7,986		2.19	
6	2	21	3	4	4	16	55,566		15.84	
6	2	11	4	4	4	16	87,846		24.06	

8.0 Appendix A

A summary of notation can be found in Table 4 below.

T	Time horizon
t	Time index $\{1,2,\ldots,T\}$
N	Number of arrivals
n	Arrival index $\{1,2,\ldots,N\}$
p_n	Probability of an arrival n
r_n	Reward of arrival n
$w_{n,k}$	Capacity of arrival n for knapsack k (elements of capacity verctor)
$\mathbf{C}_n$	Vector representing the capacity of arrival n
δ	Decision sets for n arrivals
ψ	Arrival sets for n arrival
P_ψ	Probability of the arrival set ψ
$R_{\delta,\psi}$	Reward for arrivals $\{n \mid n \in \delta \cap \psi\}$
K	Number of knapsacks that make up the capacity window
k	Knapsack index $\{1,2,\ldots,K\}$
b_k	Utilized capacity of knapsack k
$\overline{b}_k$	Maximum capacity of knapsack k
$\mathbf{S}$	State of system (vector of utilized capacity over K time periods)
$\hat{\mathbf{S}}$	State of system translated by one time period
$\mathbf{S}'_{\delta,\psi}$	State of the system resulting from arrivals $\{n \mid n \in \delta \cap \psi\}$
$V(\mathbf{S})$	Reward of system state $\mathbf{S}$
B	Set of all valid states for which each capacity constraint is not violated.
β	Number of capacity buckets.
$\widetilde{S}^{base}$	Capacity window of state S in numerical *base*.
$\hat{S}^{base}$	Capacity window of state S in numerical *base* translated one period.
$\widetilde{C}_n^{base}$	Arrival capacity of decision set δ and arrival set ψ in terms of numerical *base*.
$\widetilde{\Delta}_\delta^{base}$	Decision set representation of decision set δ in numerical *base*
$\widetilde{\Psi}_\psi^{base}$	Arrival set representation of arrival set ψ in numerical *base*

Table 4 - Summary of Notation

9.0 References

[1] Martello, S. and Toth, P. Knapsack Problems, Algorithms and Computer Implementations. John Wiley & Sons, West Sussex, England. 1990.

[2] Kleywegt, Anton J., Papastavrou, Jason D. The Dynamic and Stochastic Knapsack Problem. *Operations Research.* **46** (1998) 17-35

[3] Kleywegt, Anton J., Papastavrou, Jason D. The Dynamic and Stochastic Knapsack Problem with Random Sized Items. *Operations Research.* **49** (2001) 26-41.

[4] Bodily, Samuel E., Weatherford, Lawrence R. A Taxonomy and Research Overview of Perishable-Asset Revenue Management: Yield Management, Overbooking and Pricing. *Operations Research.* **40** (1992) 831-844.

[5] Ross, K. and Tsang, D. The Stochastic Knapsack Problem. *IEEE Transactions on Communications.* **37** (1989) 740-747.

[6] Bean, James C., John R. Birge, and Robert L. Smith. Aggregation in Dynamic Programming. *Operations Research.* **35** (1987) 215-220.

[7] Kim, David S. and Robert L. Smith An Exact Aggregation/Disaggregation Algorithm for Large Scale Markov Chains. *Naval Research Logistics.* **42** (1995) 1115-1128.

[8] T.C. Perry and J.C. Hartman. Flexible Implementation of Dynamic Programs using Numerical Representation of State Space Vectors. Industrial and systems Engineering, Technical Report, Lehigh University, 004T-008, (2004).

T	N	β	K	δ	ψ	$\delta\psi$	Full DP States	Valid DP States	Full DP Time (sec)	Valid DP Time (sec)
6	2	31	3	4	4	16	178,746		49.09	
6	2	11	5	4	4	16	966,306		287.50	
6	2	30	4	4	4	16	4,860,000		1422.09	
6	2	17	5	4	4	16	8,519,142		2516.52	
6	2	11	6	4	4	16	10,629,366		3453.53	
5	3	11	3	8	8	64	6,655		4.34	
6	3	11	3	8	8	64	7,986		5.76	
6	3	11	4	8	8	64	87,846		60.95	
6	3	13	5	8	8	64	2,227,758		1470.63	
6	3	15	5	8	8	64	4,556,250		3097.30	
7	3	10	6	8	8	64	7,000,000		5316.16	
6	4	11	4	16	16	256	87,846		140.47	
6	4	11	5	16	16	256	966,306		1609.64	
11	4	12	5	16	16	256	2,737,152		5657.44	
6	4	30	4	16	16	256	4,860,000		8201.20	

Table 3 – Results of Dynamic Stochastic Multi-Knapsack on 2.6GHz Pentium IV

6.0 Conclusion

The model has been implemented as a Microsoft Excel macro. The premise for this model is that is could be flexible an able to integrate with desktop tools on the PC to quickly solve item acceptance decisions for manufacturing lines. The introduction of the numerical representation of the capacity utilization vectors, the decision sets and the arrival sets greatly improves the looping complexity of the dynamic program. The system state, decision set and arrival nested loops can all be replaced with a single loop for each. This not only reduces the complexity of the DP, but also serves to add flexibility in the size of the capacity window that is tracked and the number of arrivals per time period without having to change the physical implementation of the DP algorithm.

The execution time is dependent on the number of states, the number of arrivals and the time horizon. Building a valid state network prior to solving the DP is shown to reduce the execution time. Extensions of this problem would include implementing future arrival pre-processing algorithms to estimate the cost-to-go function, looking into possible approximate schemes at each node to quickly determine optimal path given the arrivals present or further reduction the number of states.

7.0 Acknowledgements

I want to thank Dr. R. Storer, Dr. A. Ross and Dr. C. Pearce for their support in defining this problem and for their review. This research was in support of NSF grant....