Flexible Implementation of Dynamic Programs Using Numerical Representations of State Space Vectors

Thomas C. Perry
Joseph C. Hartman
Lehigh University

Report No. 04T-008

Flexible Implementation of Dynamic Programs using Numerical Representations of State Space Vectors

Thomas C. Perry
Agere Systems, 555 Union Boulevard, Allentown, Pennsylvania 18109

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

Vectors often represent the state space in dynamic programs. Unfortunately, changes to the vectors, such as an increase in the number of dimensions, generally requires a re-writing of the computer implementation of the dynamic programming algorithm. This paper introduces a generic technique of representing vectors as numerically equivalent numbers. This modeling approach has been applied successfully in other settings, such as the use of binary numbers (base 2) to represent computer operations. Using this technique, not only can the dynamic programming structure be made less complicated, it also allows for flexibility in system changes that do not result in re-writes of the computer implementation code. Numerous examples illustrate this technique.

Key words: dynamic programming; models; data structures

1.0 Introduction

Dynamic programs of real world systems often model the state space with a vector. Examples include, but are not limited to, multi-period perishable asset revenue management (PARM) [1], where the vector may represent utilized capacity over time; equipment replacement [2] [3] [4], where the vector may represent the number of assets in each period according to age; or water reservoir management [5] [6], where the vector may represent the amount of water in each reservoir in a given period. In each of these examples, the vector defines the state of the system. In general, transitions from one state to another are governed by (1) the decisions in a time period (such as the amount of capacity to allocate to demand in a given period in the PARM) given the costs of the decision; and (2) system constraints (such as the maximum capacity of the system in a given period in PARM) which dictate valid states and state transitions.

Implementing such a system with dynamic programming in a computer program generally involves a programming structure that is specific to the system being modeled. If any of the factors that influence these state transitions change, the underlying dynamic programming algorithm will often change. This can be as straightforward as increasing the number of times a loop is executed to as complicated as changing the number of loops and/or their ordering in the computer program. These changes greatly complicate "scenario" or "what-if" modeling because changes in system parameters often result in "re-writes" of the program.

To simplify the computer implementation of these models and to avoid re-writing the algorithm whenever system parameters change, we introduce the concept of representing the state space vector as a number. The benefit of this representation is through increased flexibility, as most changes in system parameters do not require changes to the underlying computer implementation of the dynamic program. Thus, the *same* algorithm can perform "scenario" or "what-if"

modeling. Note that this approach does not address dynamic programming's 'curse of dimensionality' [7], but rather, this issue of flexibility in implementation.

We have searched the literature and have not found any examples using this numerical technique to represent vectors in dynamic programs. Binary number systems (base 2) have been used extensively in computer applications where all mathematical operations are carried out in binary. One instance was found where a ternary numeral system (base 3) was used in self-avoiding walk problems [8]. The base 3 numbers were used to represent left, right and straight decisions in a walk.

Section 2.0 will look at a typical dynamic program structure to implement a model that uses a state vector. Section 3.0 will introduce this simple concept of representing a vector as a number. Section 4.0 will describe the potential benefits of representing the vector as a single number. Sections 5.0, 6.0 and 7.0 will illustrate this concept with examples.

2.0 Typical Dynamic Program Implementation

Using a vector to represent the state space of a dynamic program is often a practical way to define the state of the system, but can be difficult to implement in a computer program. A typical implementation results in a computer program with a 5-loop structure like the one summarized below and illustrated in Figure 1. (We assume a discrete time period in this discussion.)

A. Time Period Loop – loop over all time periods.
B. System State Loop – loop over all possible system states.
C. Decision Set Loop – loop over all possible decision sets.
D. Probabilistic Outcomes Loop – loop over all possible probabilistic outcomes.
E. Cost Function Evaluation and Constraint Loop – loop to evaluate the cost function and any constraints.

There are two disadvantages to this type of implementation. The first disadvantage is that this structure requires multiple nested loops to account for each vector element in the system state and constraint loops (Loops B & E). In addition, multiple loops are required for each possible decision and probabilistic outcome (loops C and D). This tends to complicate the implementation of the model in a computer program. The second disadvantage is that the multiple nested loops are dependent on the system characteristics being modeled. Any change in these characteristics results in a change to the looping structure, often requiring a re-write of the computer program. This severely limits using such the computer implementation to study systems with changing characteristics ("what-if" modeling). This lack of flexibility often makes the computer model un-usable in some circumstances.

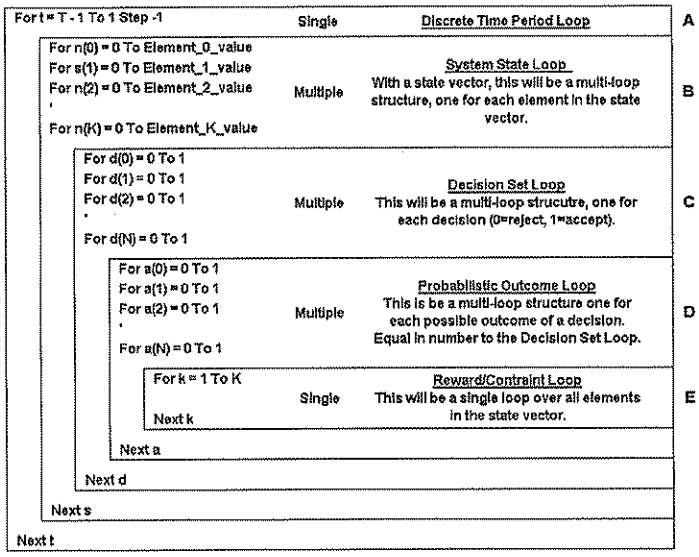

Figure 1 - DP loop structure with system state vector.

3.0 Numerical Representation of a Vector

In order to illustrate representing a vector as a number, we use the state vector:

$$\mathbf{S} = [n_1, n_2, \ldots, n_K],$$

where K is the number of elements in the state vector and n_k represents a characteristic of the system which defines the system state. This vector can be represented as a number if the set of all possible values for each element n_k, represented as E_k, are integer values and less than 10, or:

$$E_k \subseteq \{0,1,\ldots 9\} \forall k .$$

For example, a vector $[1,6,4,9]$ can be represented as the number 1649. For vectors with elements whose digits are integer values but are greater than 9, a single digit can no longer represent the value of the element in the numerical representation. This issue is resolved if we apply the concept of numerical bases.

For the previous example, each element has 10 unique possible states (integers 0 through 9), so the numerical representation is said to be in the decimal numeral system or base 10 system. Decimal numbers are what we typically use in everyday mathematics. In order to represent vectors with elements whose values have more or less than 10 unique states, numeral systems of other bases need to be considered. Higher base numbers are formed by using the digits 0 through 9 combined with using letters or other characters to represent the digits greater than 9. For this paper, only letters will be considered, so numbers greater than 9 can be represented as shown below:

$$\{0,1,2,\ldots 10,11,12,\ldots\} \rightarrow \{0,1,2,\ldots A,B,C,\ldots\}.$$

In general, the base needed to represent a vector as a number is determined from the maximum value that is possible from all the elements of the vector, or:

$$\beta = \max(E_1 \times E_2 \times \cdots \times E_K) + 1.$$

This implies that the set of integer values for each element will be a subset of the integers from 0 to $\beta - 1$, or

$$E_k \subseteq \{0,1,\ldots \beta - 1\} \forall k.$$

For example, if the elements of the state vector can take on 14 unique states, or $\beta = 14$, the vector $\mathbf{S} = [4,10,8,13]$ would be represented as the number 4A8D. (Throughout this paper, the numerical representation of the vector $\mathbf{S}$ in base β, will be represented by $\widetilde{S}^{\beta}$.)

Once a vector is represented as a base β number, it can be converted to a base 10 number. For example, if a number in base 5 is:

$$\widetilde{S}^5 = 314,$$

the corresponding base 10 number is determined as follows:

$$\widetilde{S}^{10} = 3(5^2) + 1(5^1) + 4(5^0) = 84.$$

The advantage of this conversion is that the mathematics needed to transition from one state to another can be carried out in base 10, as is common. However, even though the mathematics are in base 10, the physical representation of the system in base β is preserved. This is detailed further in the next sections.

3.1 State Transitions

A system transitions from state $\mathbf{S}$ to another state, $\mathbf{S}'$, at each discrete time period. The properties of the state transition are dependent on the specific details of system being modeled. But, in general, two fundamental operations can be applied to the numerical representation of the state vectors:

1. The addition (subtraction) of the system parameter to (from) the state vector.
2. The elements in the state vector are shifted right or shifted left by one position.

Both of these operations are fundamental ways to manage the transition of the state vectors that are represented by numbers as described in more detail below.

3.1.1 Addition & Subtraction

For a given time period, decisions are evaluated for each state in the system. Each decision often requires the addition (subtraction) of a system parameter that is represented by the state vector, to (from) the state vector. For example, if the state vector represents the number of assets in a system, acceptance of another asset would require the addition of another unit to the state vector.

Although not difficult, addition or subtraction in a base other than base 10 is not straightforward. Therefore, it would be easier to describe all vectors in base 10 equivalent numbers. Adding or subtracting vectors represented in base 10 is merely performing normal addition or subtraction. Consider the vectors $\mathbf{S}_1$ and $\mathbf{S}_2$ in Table 1 below.

	Base = 4	Base = 10
$\mathbf{S}_1 = [2,3,0,1,0]$	$\widetilde{S}_1^4 = 23010$	$\widetilde{S}_1^{10} = 708$
$\mathbf{S}_2 = [1,0,1,0,0]$	$\widetilde{S}_2^4 = 10100$	$\widetilde{S}_2^{10} = 272$
$\mathbf{S}_1 + \mathbf{S}_2 = [3,3,1,1,0]$	$\widetilde{S}_1^4 + \widetilde{S}_2^4 = 33110$	$\widetilde{S}_1^{10} + \widetilde{S}_2^{10} = 980$

Table 1 – Vector addition in base 4 and the equivalent base 10

From Table 1, it is clear that 980 in base 10 is equivalent to 33110 in base 4. Performing the addition in base 10 carries along the physical representation of the vector in the base 4 system. Thus, we can carry out all calculations in base 10 instead of using the system parameters explicitly.

3.1.2 Shift Right, Shift Left

Often with a transition from one time period to the next, the state vector needs to be transitioned as well. This is generally true when each element in the vector is identified with a time period. Figure 2 represents a vector with four elements (represented in Figure 2(A)). If the vector elements are transferred to the right (shift right), the last element is discarded and the first element position becomes empty (represented in Figure 2(B)). Similarly, if the vector elements are transferred to the left (shift left), the first element is discarded and the last element position becomes empty (represented in Figure 2(C)).

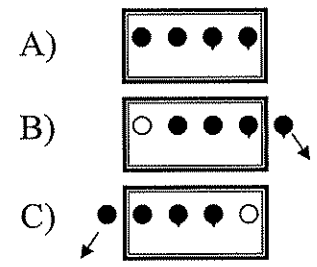

Figure 2 - Illustrates shifting vector elements either left or right.

Intuitively, these transitions are straightforward. If the vectors are represented in base 10, a shift right can be accomplished through multiplication of the number by 10 and removing the leftmost digit. For example, $4357 \cdot 10 = 43570$ and removing the leftmost digit results in 3570 and, therefore, the transition $4357 \rightarrow 3570$ is completed. Similarly, a shift left is accomplished through the division of the number by 10 and removing the fractional part of the number.

These same concepts can be applied to number bases other than base 10. The shift left translation process is nothing more than multiplying the base 10 representation of the vector by the numerical base that is being represented. After this multiplication, the leading (most significant digit) in the base being represented by the base 10 number needs to be removed. For example, let a state vector in base, $\beta = 21$, with the number of vector elements, $K = 5$ be:

$$\mathbf{S} = [1,2,3,4,5].$$

The equivalent base 10 numerical representation of this capacity utilization vector is:

$$\widetilde{S}^{10} = 214415.$$

A shift left translation is performed by multiplying this number representation by base β, or:

$$\beta \widetilde{S}^{10} = 21(214415) = 4502715,$$

from which the leading digit is stripped. In order to accomplish this, the leading digital must be represented in base β equivalent. The leading digit can be determined from the base 10 number by:

$$\text{leading digit} = \text{int}\left(\frac{\widetilde{S}^{10}}{\beta^{(K-1)}}\right).$$

The leading digit in this case is:

$$\text{leading digit} = \text{int}\left(\frac{214415}{21^{(5-1)}}\right) = 1,$$

and corresponds to an equivalent base 10 number of:

$$1 \cdot \beta^{K} = 1 \cdot 21^{5} = 4084101,$$

since there are $K = 5$ elements and β^{K} is the base 10 value of this most significant digit in base β. Subtracting this number from the one obtained above, is equivalent to removing the most significant digit in base β and results in:

$$\widetilde{S}^{10} = 4502715 - 4084101 = 418614.$$

This is equivalent to:

$$\widetilde{S}^4 = 23450,$$

or:

$$\mathbf{S} = [2,3,4,5,0],$$

which represents the initial vector whose elements have been shifted left.

In general, a shift left transition of a vector of base β represented as a base 10 number can be generalized as follows:

$$\widetilde{S}^{10} = \widetilde{S}^{10}\beta - \text{int}\left(\frac{\widetilde{S}^{10}}{\beta^{(K-1)}}\right)\beta^K,$$

where K is the number of elements in the vector. This can be simplified further by using the Modulo function available in most programming languages, or:

$$\widetilde{S}^{10} = \widetilde{S}^{10}\beta \bmod \beta^K.$$

In a similar fashion, the shift right transition process is defined as:

$$\widetilde{S}^{10} = \text{int}\left(\frac{\widetilde{S}^{10}}{\beta}\right).$$

3.2 Decision/Probabilistic Outcome Sets

At each time period, each decision in the decision set needs to be either accepted or rejected. These decisions are often implemented with a binary representation (0 for reject or 1 for accept) in a dynamic program. If there are multiple decisions, multiple nested loops in the dynamic program account for all of the possible decision combinations. In general, there are 2^N possible combinations where N is the number of decisions. Just like the state vector, a decision vector can be formulated that represents a possible decision combination. If there are $N = 3$ possible decisions, an example decision vector may be:

$$\mathbf{D} = [0,1,0],$$

where the decision represented by the first and third elements are "reject" and the decision represented by the second is "accept". Since there are two possible states, $\{0,1\}$, the number to represent this decision vector would be in base $\beta = 2$.

Decisions can be pre-processed prior to looping through each state in a dynamic program. In this way, a look-up table can be generated based on the decision combinations in base 10 with the cost and state transition information of the decision.

Since there is a one-to-one relationship between the decisions and their probable outcomes, the same concept can be applied to form an outcomes vector. This vector can be represented as a base 10 number and a look-up table can be generated based on the outcome combinations with the cost and state transition information of the outcome.

3.3 Vector Element Constraints

Since a vector element has a finite number of possible states, it is often useful to be able to quickly tell if a vector element has reached its maximum possible value when an addition is performed. For example, if the vector element represents capacity utilization, anytime this element exceeds 100 (capacity is 100% utilized), the capacity constraint is violated. Consider the following vector which represents capacity utilization:

$$\mathbf{S} = [80,90,40],$$

where the maximum value of any element is 100. Assume an arrival combination consumes the following capacity:

$$\mathbf{C} = [10,20,40].$$

If these two vectors are added, the second element would result in a capacity utilization greater than 100, which is infeasible.

The numerical representation of the vector can provide an efficient way to determine if any of the elements exceed their maximum values. When a state transition occurs which involves adding vector elements, a feasible element value only occurs when there is no algebraic carry in the addition of the equivalent vector numbers for any digit.

Let $X(\alpha)$ represent the right-hand portion of the state vector number starting at the least significant digit and containing α digits:

$$X(\alpha) = \tilde{S}^{10} - \text{int}\left(\frac{\tilde{S}^{10}}{\beta^{\alpha}}\right)\beta^{\alpha} \quad \forall \alpha \in \{1,\ldots,K\}.$$

Let $Y(\alpha)$ represent the right-hand portion of the arrival combination vector number starting at the least significant digit and containing α digits:

$$Y(\alpha) = \tilde{C}^{10} - \text{int}\left(\frac{\tilde{C}^{10}}{\beta^{\alpha}}\right)\beta^{\alpha} \quad \forall \alpha \in \{1,\ldots,K\}.$$

Let $Z(\alpha)$ represent the right-hand portion of the resulting vector number starting at the least significant digit and containing α digits:

$$Z(\alpha) = \left(\tilde{S}^{10} + \tilde{C}^{10}\right) - \text{int}\left(\frac{\left(\tilde{S}^{10} + \tilde{C}^{10}\right)}{\beta^{\alpha}}\right)\beta^{\alpha} \quad \forall \alpha \in \{1,\ldots,K\}.$$

There are no algebraic carries if $X(\alpha)+Y(\alpha)=Z(\alpha)$ for all $\alpha \in \{1,\ldots,K\}$. This can be simplified further by using the Modulo function, or:

$$X(\alpha) = \tilde{S}^{10} \bmod \beta^{\alpha} \quad \forall \alpha \in \{1\ldots K\},$$

$$Y(\alpha) = \tilde{C}^{10} \bmod \beta^{\alpha} \quad \forall \alpha \in \{1,\ldots,K\},$$

and

$$Z(\alpha) = \left(\tilde{S}^{10} + \tilde{C}^{10}\right) \bmod \beta^{\alpha} \quad \forall \alpha \in \{1,\ldots,K\}.$$

4.0 Benefits of Representing a Vector as a Number

As described in section 2.0, there are two disadvantages in the typical implementation of a dynamic program with a multi-dimensional state vector in a computer program. They are (1) the need for multiple loops and (2) the inflexibility to system changes. These issue can be mitigated by representing the vectors as numbers, carrying out all operations in numerically equivalent base 10 mathematics.

Using numerically equivalent representations of the vectors results in a simplified dynamic programming loop structure. Loops B and E in Figure 1 typically involve multiple nested loops, one for each element in the vectors, which are eliminated when the vector is represented as a number. A single loop structure can then handle all of the physical combinations of the vector elements. This is also true when the decision and outcome vectors are represented as numbers. The multiple nested loop structures typically needed in Loops C and D in Figure 1 are reduced to a single loop each. Again, the single loop structure can handle all of the physical combinations of the decisions and outcomes. The simplified dynamic programming structure with a state vector represented as a number is shown in Figure 3 below.

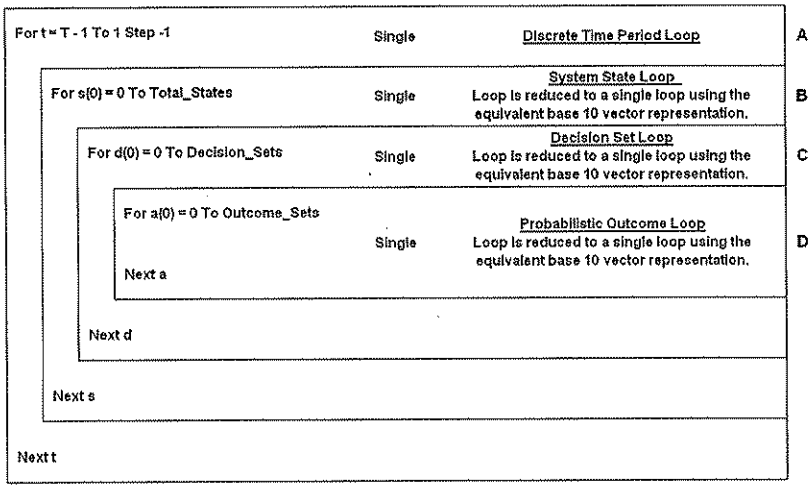

Figure 3 - Reduced DP Loop Structure

Also, since the vectors are represented as numbers and not represented in the dedicated loops of the computer program, it is easy to accommodate system changes. This introduced flexibility comes without the need to modify the looping structure of the computer program. Changes in the vectors only change the numerical representation, which means that only the ending or stopping value of the loop would need to be modified. This change is easily handled by treating the loop bound as variable in the computer program, which can be input at run time. The key is that the looping structure does not change.

Note that these benefits, however, do not address the 'curse of dimensionality' [7] often encountered with dynamic programs. This technique does not necessarily reduce the number of overall calculations and may in fact increase the number of calculations. Other techniques such as state aggregation [9] and approximation schemes [10] [11] to reduce the number of states and calculations must still be considered for large-scale dynamic programs. To illustrate the application of these concepts, the computer program implementation of several applications are described in the following sections.

5.0 Multi-Period Perishable Asset Revenue Management Problem

Perishable asset revenue management problems often involve the acceptance of items that arrive with known sizes (weight) and rewards (cost) (Weatherford and Bodily [1]). Kleywegt and Papastravrou [12] [13] extend this to arrivals with unknown cost and unknown rewards. Consider a multi-period problem where items, representing orders, arrive and if accepted, are placed into 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 which may be accepted, if capacity is available, or rejected. If an item arrives but there is no capacity, it is automatically rejected. The details of this model are described in Perry and Hartman [14].

Such a system can be modeled in discrete time. 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. The state space can be represented by the multi-period vector:

$$\mathbf{S}_t = [n_{t,1}, n_{t,2}, \ldots, n_{t,K}],$$

where K is the number of knapsack in the state vector and $n_{t,k}$ represents utilized capacity in time period t of knapsack k. At each time period a decision is made to accept or reject orders and the system transitions to another state, defining the new utilization levels. The capacity needed by the order is represented by the arrival capacity vector:

$$\mathbf{C} = [c_1, c_2, \ldots, c_K],$$

where c_k is the capacity needed by an order in period k. As with any perishable asset, the capacity is either consumed or left underutilized as the system transitions in time. Given our original state of the system, the state of the system at the next decision epoch is:

$$\mathbf{S}_{t+1} = [n_{t+1,1}, n_{t+1,2}, \ldots, n_{t+1,K}],$$

where the capacity utilization of each knapsack is:

$$n_{t+1,k} = n_{t,k+1} + c_k \quad \forall k = 1, \ldots, K-1,$$

and:

$$n_{t+1,K} + c_K = 0.$$

If the utilized capacity of a system is represented by $K = 5$ knapsacks:

$$\mathbf{S}_t = [8,6,9,5,2],$$

and the capacity of an order is:

$$\mathbf{C} = [1,1,2,1,2].$$

The next state would be:

$$\mathbf{S}_t = [7,10,7,3,2].$$

If the capacity utilization levels can take on integer values in multiples of 10, each knapsack can take on 11 values $\{0,1,2,\ldots 10\}$. A base 11 ($\beta = 11$) number could be used to represent the state vector as a number. The state vector above is:

$$\widetilde{S}_t^{11} = 86952,$$

or in base 10:

$$\widetilde{S}_t^{10} = 126260,$$

and the order capacity vector is:

$$\widetilde{C}^{11} = 11212,$$

or in base 10:

$$\widetilde{C}^{10} = 16227.$$

To perform the state transition in these numerically equivalent numbers would involve a shift left, representing the capacity that is shifted in time. Once the shift left is performed, the numerically equivalent order capacity can be added. From sections 3.1.1 and 3.1.2, this can be represented as:

$$\widetilde{S}_{t+1}^{10} = \left(\widetilde{S}_t^{10} \beta - \text{int}\left(\frac{\widetilde{S}^{10}}{\beta^{(K-1)}} \right) \beta^K \right) + \widetilde{C}^{10},$$

or:

$$\widetilde{S}_{t+1}^{10} = \left(126260 \cdot 11 - \text{int}\left(\frac{126260}{11^4} \right) \cdot 11^5 \right) + 16227,$$

or:

$$\widetilde{S}_{t+1}^{10} = 116679.$$

This is numerically equivalent in base 11 to:

$$\widetilde{S}_{t+1}^{11} = 7\text{A}732,$$

which represents the vector:

$$\mathbf{S}_{t+1} = [7,10,7,3,2],$$

which is clearly equivalent to the original vector problem.

This model can also benefit from the numerical representation of the decisions and probabilistic arrivals. Each can be represented as numerically equivalent base 2 numbers as outlined in section 3.2. In addition, 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 or all knapsacks. This can be accomplished with the technique described in section 3.3 where each vector element (knapsack utilization) would be evaluated for an algebraic carry. If one exists, the capacity constraints are violated.

The state vector for this type of model is governed by the number of elements and the number of values each element can take on to represent capacity utilization. If either of these would change, loops B and E in Figure 1 would need to be updated in the computer implementation. Using a numerically equivalent representation, this is not needed. For example, consider a 5 period problem in which 5 knapsacks are represented in a state vector. A 5 nested loop structure would be needed without the numerically equivalent representation, one loop for each vector element. But, with the numerically equivalent representation, only the number of states would change. The number of states is β^K, where β is the number of values the vector element can take on and K is the number of knapsacks. If the capacity of the knapsack is represented by 6 integers $\{0,1,2,\ldots 5\}$ (evenly spaced capacity intervals) then, for this example, there would be 6^5 or 7,776 states. State 0 represents all knapsacks empty and state 7,775 represents all knapsacks full. All numbers in between represent various combinations of the knapsack states, empty to full. If we decided to represent the capacity of the knapsack by 10 evenly sized buckets, the total number of states would just change to 10^5 or 100,000. All values between 0 and 100,000 represent the various combinations of the knapsack states, empty to full. To incorporate this into the computer program, only the loop increments need to be updated, not the number of loops. The structure of a typical implementation also changes if the number of arrivals change. A comparison of how changes affect both the typical implementation and the implementation using numerical representation is shown in Table 2 below.

Model Parameter	Solution in Typical DP	Solution in DP with Numerical Representation
Time Periods	State loop stop point is changed. No change to loop structure.	State loop stop point is changed. No change to loop structure.
Number of Arrivals	Increase (or decrease) in the number of nested loops which iterate through all arrival combinations. **Requires program changes.**	Decision loop stop point is changed. No change to loop structure.
Number of Knapsacks	Increase (or decrease) in the number of nested loops which iterate through all system state combinations. **Requires program changes.**	System state loop stop point is changed. No change to loop structure.

Capacity Element Value	The System state loops (nested) increment is increased (or decreased). No change to the loop structure.	System state loop stop point is changed. No change to loop structure.
Constraint	Loop through all knapsacks checking capacity.	Check for algebraic carry.

Table 2 - The comparison of computer programs changes for PARM problems

6.0 Parallel Replacement Problem

Parallel replacement involves determining when assets among a group of similar assets are to be replaced over time. The system can be described by the number of pieces of equipment categorized by their age. This information can be represented in the state vector $\mathbf{N}$ as:

$$\mathbf{N} = [n_1, n_2, \ldots, n_A],$$

where n_i is the number of assets of age i and A is the maximum age of an asset. At each time period, a decision is made to keep or replace a set of assets in a given age group. If the decision is to replace, then the new assets (age one at the end of the next period) are defined:

$$n_1 = r_A + \sum_{i=1}^{A-1} r_i ,$$

where r_i is the number of assets to replace from age group i (the asset at their maximum age must be replaced). The assets that are kept age one period, or:

$$n_{i+1} = n_i - r_i \quad \forall i = 1, 2, \ldots, A-1 .$$

If there is a budget constraint where at most 15 assets can be replaced in each time period and the maximum age of any asset is 5 years, a typical state vector at time t is:

$$\mathbf{N}_t = [12, 8, 15, 10, 10].$$

If the decision to replace at time t is $r_3 = 1$, $r_4 = 2$ and $r_5 = 10$, the replacement decision is represented as:

$$\mathbf{R}_t = [r_0, r_1, r_2, r_3, r_4],$$

or in this case:

$$\mathbf{R}_t = [0, 0, 1, 2, 10].$$

. The resulting state of the system after the replacement of the assets would be:

$$\mathbf{N}_{t+1} = [13,12,8,14,8].$$

Both the vector representing the state of the system and the vector representing the asset replacement decisions can be expressed in a numerically equivalent number representation. Since 15 is the total number of assets in any given age period, each element of the state vector can then take on 16 possible states, $\{0,1,2,\ldots,15\}$. Therefore, a numerical base of 16, or $\beta = 16$, is used. The state vector:

$$\mathbf{N}_t = [12,8,15,10,10],$$

would translate to a base 16 number of:

$$\tilde{N}_t^{16} = C8FAA,$$

or a base 10 number of:

$$\tilde{N}_t^{10} = 823210.$$

The replacement decision vector:

$$\mathbf{R}_t = [0,0,1,2,10],$$

would translate to a base 16 number of:

$$\tilde{R}_t^{16} = 12\text{A}$$

or a base 10 number of:

$$\tilde{R}_t^{10} = 298.$$

The number of assets that would need to be purchased in the current time period (would age one period at the end of the time period) can be represented by the vector:

$$\mathbf{P}_t = [13,0,0,0,0],$$

where the first element is determined by the total assets replaced and would translate to a base 16 number of:

$$\tilde{P}_t^{16} = \text{D0000},$$

or a base 10 number of:

$$\widetilde{P}_t^{10} = 851968,$$

The state transition in numerically equivalent base 10 numbers is performed by first subtracting the replacement decision vector $\widetilde{R}_t^{10}$ from the state vector $\widetilde{N}_t^{10}$, perform a shift right and finally adding the number purchased assets $\widetilde{P}_t^{10}$, or:

$$\widetilde{N}_{t+1}^{10} = \text{int}\left(\frac{\widetilde{N}_t^{10} - \widetilde{R}^{10}}{\beta}\right) + \widetilde{P}_t^{10} = \text{int}\left(\frac{823210 - 298}{16}\right) + 851968 = 903400.$$

The base 16 equivalent number would be:

$$\widetilde{N}_{t+1}^{16} = \text{DC8E8},$$

which is equivalent to the vector:

$$\mathbf{N}_{t+1} = [13,12,8,14,8].$$

For parallel replacement problems, increasing the maximum age of an asset would typically require a change to the computer program. If the maximum age is increased, the number of multiple nested loops would need to be increased as well. With the numerical representation, the changes in age would just change the stop limit on the state loop. For example, considering the previous example with the budget constraint of 15 possible assets per age group and a maximum age of 5 years, the number of possible states in the numerical representation is N^A, or:

$$15^5 = 759{,}375.$$

If the maximum age were increased to 6 years, the number of possible state increases to 11,390,625. Only the loop stop increment would need to change in the computer program. A comparison of how changes affect both the typical DP and the DP using numerical representation is shown in Table 3 below.

Model Parameters	Solution in Typical DP	Solution in DP with Numerical Representation
Time Periods	State loop stop point is changed. No change to loop structure.	State loop stop point is changed. No change to loop structure.
Number Assets	Decision loop stop point is changed for each age grouping loop. No change to loop structure.	Decision loop stop point is changed. No change to loop structure.
Maximum Age	Increase (or decrease) the number of nested loops which iterate through all age groupings. **Requires program changes.**	System state loop stop point is changed. No change to loop structure.

Constraint	Test combination of assets to replace against budget constraint.	Test combination of assets to replace against budget constraint.

Table 3 - The comparison of computer programs changes for Parallel Replacement problems

This problem highlights a potential issue when modeling the state space vector as a number. In our previous example, if there was no budget constraint and the number of assets required over time was constant (55 assets), then we would require a state space with 55^5 elements. Clearly with an increase in the maximum number of assets or the maximum age, this can be problematic. We discuss this further in section 8.0.

7.0 Water Reservoir Management

Water reservoir management problems are generally concerned with determining the amount of water to release from each reservoir in a system of reservoirs in order to satisfy some performance measure over time, such as profit from electricity generation. These are generally modeled with dynamic programming with the state vector tracking the water level in each reservoir over time. Define the amount of water in reservoir i as V_i. Each reservoir has a maximum capacity and initial level of water at time zero. The amount of rain captured each period is probabilistic, r_i and the decision is how much water to release each period. The transitions depend on how the reservoirs are connected. For example, reservoir V_1 may feed reservoir V_2. The general state transition equation is:

$$V_i^{t+1} = V_i^t - \sum_j T_{i,j}^t + \sum_j T_{j,i}^t + r_i,$$

where V_i^t is the initial amount of water, $T_{i,j}^t$ is the amount transferred from reservoir i into j and r_i is the amount of rain (or evaporation).

With several reservoirs, the state vector $\mathbf{W}$ is given by:

$$\mathbf{W} = [V_1, V_2, \ldots, V_R],$$

where V_i is the volume of water in reservoir i and the total number of reservoirs is R. In a similar fashion to the parallel replacement problem, it can be shown that each of the operations in a reservoir management problem can be carried out in a base 10 numerical representation. There are three general transitions that can cause the change of state, they are:

1. Transfer water out to generating electricity and/or to another reservoir
2. Transfer water in from another reservoir
3. Addition of water from rain

These operations involve only addition and subtraction. The elements of the state vector can represent the utilized capacity and be aggregated into integer increments of capacity. Therefore,

the base for the numerical representation of the state vector would be equal to the number of integer elements of capacity levels chosen. If there were 21 capacity units used to represent capacity (0 units is empty and 21 units is full), the state vector may look like:

$$\mathbf{W} = [8, J, A].$$

This in turn would translate to a base 21 number as

$$\widetilde{W}^{21} = 8\text{JA}.$$

To add or subtract water from any of these reservoirs simply involves forming a vector to represent what is added and one to represent what is subtracted. If 2 capacity levels are added to reservoir 3, the base 21 equivalent number would be 003 or 3. If 5 units of capacity are taken away from each reservoir, the base 21 equivalent number would be 555. The equivalent operations in base 21 are shown below:

$$8\text{JA} + 3 - 555 = 3\text{E}8,$$

or in base 10 it would be carried out as:

$$3937 + 3 - 2315 = 1625.$$

It can be shown that 1625 base 10 is numerically equivalent to 3E8 base 21.

A key advantage of using this technique is that instead of using individual loops that operate over each reservoir, a single loop structure can be used to operate over all numeric representations of states. This structure would make it very easy to add or remove reservoirs to the model without having to change the looping in the DP program. However, this problem is unique with respect to the other two that were presented in sections 5.0 and 6.0 because reservoirs involve physical connections (rivers). A complication to the computer implementation is the ability to study scenarios in which these physical connects change (reservoir placement).

In a traditional dynamic program, a physical connection change results in a computer implementation change. In the general state transition equation given above, the matrix $T_{i,j}$ is used to represent the flow of water from reservoir i to reservoir j. In the most general form, this matrix can allow water to flow from one to any other reservoir. However, this is often impractical for reservoir systems. Since a river is often what connects at most two reservoirs, it is common to restrict the flow of a reservoir to only one other reservoir. This results in a matrix with only one non-zero element per row & column which can be represented as a vector. The "flow out" vector can be determined by summing the rows of the $T_{i,j}$ matrix, or:

$$\mathbf{Fout} = [fout_1, fout_2, \ldots, fout_R],$$

where $fout_i$ is the amount of wafer flowing out of reservoir i. The "flow in" vector can be determined by summing the columns of the $T_{i,j}$ matrix, or:

$$\mathbf{Fin} = [fin_1, fin_2, \ldots, fin_R],$$

where fin_j is the amount of wafer flowing into reservoir j. Since the model will look at all possible combinations of the "flow out" vector, **Fout**, a way is needed to generate the "flow in" vector, **Fin**, given the interconnection scheme of the reservoirs. Consider the connection vector used to represent the interconnect scheme of the reservoirs:

$$\mathbf{C} = [c_1, c_2, \ldots, c_R],$$

where c_i represents the reservoir downstream to reservoir i. The elements of the "flow in" vector can be determined by:

$$fin_i = \sum_{\{j: C_j = i\}} fout_j \quad \forall i.$$

Using this technique along with the numerical representation of the vectors, allows for a single looping structure to be used for all possible water release combinations and various reservoir connection schemes. This can be extremely useful when considering changes to the reservoir system (i.e. adding a reservoir).

A comparison of how changes affect both the typical computer implementation and the implementation using numerical representation is shown in Table 4 below.

Model Parameters	Solution in Typical DP	Solution in DP with Numerical Representation
Time Periods	State loop stop point is changed. No change to loop structure.	State loop stop point is changed. No change to loop structure.
Number of Reservoirs	Increase (or decrease) the number of nested state loops which iterate through all reservoir combinations. **Requires program changes.**	Decision loop stop point is changed. No change to loop structure.
Change the interconnection of reservoirs	Change the logic of the program. **Requires program changes.**	Change the connection vector which is an input to the routine. Requires no changes to computer implementation.
Capacity Elements	Decision loop stop point is changed for each reservoir loop. No change to loop structure.	System state loop stop point is changed. No change to loop structure.
Constraint	Loop through all knapsacks checking capacity.	Check for algebraic carry.

Table 4 - The comparison of computer programs changes for Wafer Reservoir Management problems

8.0 Summary

We have shown several advantages to representing state vectors, decision vectors and arrival set vectors as equivalent numbers of the appropriate base and carrying out all operations in a base 10 equivalent. The major advantage of rendering vectors as equivalent numbers is to simplify the computer implementation of dynamic programs. By providing a simplified computer implementation, a key advantage results: The simplified program structure increases the flexibility of the computer program because even with changes that affect the number of elements in the state vectors or the number of decisions, no underlying loop structures are changed. This is a tremendous benefit when performing "what-if" scenarios which is often one of the main reasons for developing the computer implementation.

This may help in the future implementation of dynamic programs, which are often not implemented due to their structure which is specific to the problem being studied. This is why dynamic programs are often referred to as art and accounts for the lack of dynamic programming software. With this numerical representation, implementation of these types of dynamic programs may be possible and may provide flexibility in modeling "scenarios" when previously re-writes of the computer implementation was necessary and time consuming.

However, this technique does not eliminate the curse of dimensionality often associated with the evaluation large state spaces. Table 5 summarizes the number of state combinations for the three problems outlined in this paper. The state and decision combinations quickly grow with small increases in the system parameters. For example, parallel replacement problems can involve 1000 or more assets with a maximum age of 5 to 10 years. Clearly the state and decision combinations for this size problem become substantial. Thus, the modeling ease and flexibility introduced comes with a tradeoff.

Problem Type	Complications	States	Decisions
PARM	β - Capacity elements K - Knapsacks	β^K	2^N
Parallel Replacement	N - Number of assets A - Maximum asset age	N^A	N^A
Water Reservoir Management	β - Capacity elements K - Reservoirs	β^K	β^K

Table 5 - Typical state and decision combinations

Future research will focus on combining techniques which address the curse of dimensionality, such as state aggregation and/or approximation schemes, with the numerical representation of the state space. The goal is to develop dynamic programs which can solve large problem instances but retain programming flexibility.

9.0 Acknowledgements

This research was supported in part by NSF grants DMI-0121395 and DMI-9984891.

10.0 References

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

[2] W.J. Hopp, P.C. Jones, and J.L. Zydiak, A Further Note on Parallel Machine Replacement, *Nav Res Logistics* 40 (1993), 575-579.

[3] S. Rajagopalan, Capacity Expansion and equipment Replacement: A Unified Approach, *Operations Research* 46(6) (1998), 846-857.

[4] J.C. Hartman, The Parallel Replacement Problem with Demand and Capital Budgeting Constraints, *Nav Res Logistics* 47 (2000).

[5] J.R. Stedinger, B.F. Sule, and D.P. Loucks. Stochastic Dynamic Programming models for Reservoir Operation Optimization. *Water Resources Research* 20 (1984), 1499-1505.

[6] S. Yakowitz, Dynamic Programming Applications in Water Resources, *Water Resources Research* 18 (1982) 673-696.

[7] R.E. Bellman. *Dynamic Programming.* Princeton University Press, Priceton, NJ, 1957.

[8] B. Hayes. How to Avoid Yourself. *American Scientist* 86 (1998) 314-319.

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

[10] D.P. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[11] D. Bertsimas and R. Demir. An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science*, 48(4) (2002), 550-565.

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

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

[14] T.C. Perry, J.C. Hartman, Allocating Manufacturing Capacity by Solving a Dynamic Stochastic Multi-Knapsack Problem. Industrial and Systems Engineering, Lehigh University, Technical Report 004T-009 (2004).