Control and Cybernetics

vol. **35** (2006) No. 3

Approximating the solution of a dynamic, stochastic multiple knapsack problem

by

Joseph C. Hartman[1] and Thomas C. Perry[2]

[1]Industrial and Systems Engineering, Lehigh University
Bethlehem, Pennsylvania, USA

[2]Agere Systems
Allentown, Pennsylvania, USA

Abstract: We model an environment where orders arrive probabilistically over time, with their revenues and capacity requirements becoming known upon arrival. The decision is whether to accept an order, receiving a reward and reserving capacity, or reject an order, freeing capacity for possible future arrivals. We model the dynamic, stochastic multiple knapsack problem (DSMKP) with stochastic dynamic programming (SDP). Multiple knapsacks are used as orders may stay in the system for multiple periods. As the state space grows exponentially in the number of knapsacks and the number of possible orders per period, we utilize linear programming and duality to quickly approximate the end-of-horizon values for the SDP. This helps mitigate end-of-study effects when solving the SDP directly, allowing for the solution of larger problems and leading to increased quality in solutions.

Keywords: stochastic dynamic programming, approximate dynamic programming, linear programming, duality.

1. Introduction

The use of approximation techniques in solving large scale dynamic programs has become increasingly prolific to combat the curse of dimensionality (Bellman, 1957) in that the number of states to be evaluated over time grows exponentially in the number of state space parameters. Bellman (1957) noted a variety of techniques, including the use of Lagrangian relaxation and state aggregation, in order to promote the use of dynamic programming to solve a variety of problems. Today, the number of techniques used to solve these large-scale problems are numerous and as de Farias and Van Roy (2003) note, that as with working with dynamic programs in general, there has been much "art" in their development.

Research has been extensive into the use of interpolation, including polynomial (Philbrick and Kitanidis, 2001), spline (Johnson et al., 1993), and neural

network (Bertsekas and Tsitsiklis, 1996) methods. A sparse representation of the state space is utilized in order to estimate the value, or cost-to-go, function and the solution to this easier problem is then mapped to the true problem with interpolation methods.

Powell has been very successful in solving large-scale dynamic allocation (often fleet management) problems with the use of approximate dynamic programming (Powell, Carvalho, 1998; Shapiro, Powell, Simao, 2002; Godfrey, Powell, 2002; Topaloglu, Powell, 2006). His research group utilizes a number of methods, including simulation and duality from the solution of network flows, in order to generate estimates of value functions. An interesting aspect of his work is that given the shape of the value function (i.e. concave) and point estimates of the slope, then the value function itself can be updated with each new estimate (Godfrey and Powell, 2002). Our approach is motivated by these methods.

There are numerous other examples of specialized approximation algorithms. For example, Bertsimas and Demir (2002), use successive solutions of linear programming relaxations to approximate the solution of large-scale integer multiple knapsack problems. A more general approach comes from de Farias and Van Roy (2003) on approximating the linear programming formulation of a dynamic program, which leads to an exponential number of variables and constraints.

In recent research, we have modeled a dynamic, stochastic multiple knapsack problem (DSMKP) with dynamic programming to solve a capacity reservation problem applied in a manufacturing setting (Perry and Hartman, 2004). Specifically, we model the system according to the amount of utilized capacity in each period over some study horizon. Orders arrive periodically (according to a stochastic process) and, if accepted, utilize resources over a given period of time. A reward is received for accepting an order. As there is limited capacity (known) in the system that cannot be exceeded, the decision-maker must trade-off accepting an order (and reserving capacity) for an immediate reward vs. holding capacity for future order arrivals.

The problem is difficult to solve as the number of decisions grows exponentially in the number of arrivals. The state space grows in the number of periods of capacity that can be reserved and the discretization of the capacity. The stochastic dynamic program (SDP) can only be solved over a limited number of periods due to these issues. (In our experience, it takes nearly six hours to solve a four period problem assuming minimal data for an industrial application: four quarters of capacity can be reserved, tracked according to four percent increments. This assumes that we are solving the model from a given initial state and evaluating all reachable states over the decision horizon.)

As we can only solve the SDP over a few periods, approximate dynamic programming techniques can be used to extend the effective horizon of the problem being studied, thus mitigating end-of-horizon effects. A typical dynamic programming implementation assumes state values at the end of the horizon (boundary conditions) are zero and the recursion is solved backwards to the initial state for an optimal solution. This is fine for a finite horizon problem, but

can lead to incorrect time zero decisions if the horizon is too short when approximating an infinite horizon problem. In our approach, we solve an approximate dynamic program over a number of periods and utilize the approximate state space values as the boundary condition values in the full SDP recursion. This is critical as an assumption of zero for each state value provides no differentiation between states while the approximation method populates the states with non-zero values. This can then lead to optimal time zero decisions with shorter horizon (SDP) problems or allow for the solution of longer horizon problems more readily.

In this paper, we utilize a linear programming formulation of the problem and illustrate how the dual variables associated with capacity constraints can be used to approximate the end-of-study state values, as opposed to solving a linear program for each reachable state. This information drastically reduces the number of linear programs that must be solved when approximating the reachable state values, as we only need to solve a limited number of linear programs. While our results are only empirical, we believe them to be promising.

The paper proceeds as follows: In the next section, we introduce the stochastic dynamic programming formulation and discuss its state space growth over the study horizon. In Section 3, we introduce the linear programming approximation and discuss how it can be utilized to estimate the cost-to-go function. In Section 5, we illustrate its use on a large-scale decision problem. We then conclude and offer suggestions for future research.

2. Stochastic dynamic programming formulation

We formulate our problem with stochastic dynamic programming. At the beginning of each period, the state of the system is defined by the amount of utilized capacity which, in turn, defines the amount of available capacity for future orders. Orders with known revenues and capacity requirements arrive according to a stochastic process. These can be rejected or accepted given the capacity constraints. (Capacity is assumed to be known and to be a hard constraint.) The process repeats at each period, with new (future) capacity being made available and orders in the previous period exiting the system. We formally define the dynamic program as follows:

Define B_t as the capacity of the system in time period t and let b_t denote the amount of capacity utilized in period t due to accepted orders from previous periods. Thus, the state of the system at time t is defined as:

$$S_t = [b_t, b_{t+1}, \ldots, b_{t+K-1}], \quad t = 1, 2, \ldots, T, \tag{1}$$

such that K periods of capacity are tracked over time. For notational purposes, we refer to $S_t(j)$ as the j-th component of S_t, which equals the amount of utilized capacity in period $t + j - 1$. Note that the capacity constraints require that:

$$b_t \leq B_t, \quad t = 1, 2, \ldots, T. \tag{2}$$

A set of orders $\psi \in \Psi_t$ arrives in period t with probability $p_t(\psi)$. It is assumed that the set Ψ_t consists of finitely many elements and that the random variables generating the orders are stochastically independent such that the process governing ψ_{t+1} is stochastically independent of the process governing ψ_t.

An order in ψ_t can be accepted or rejected, subject to capacity constraints. Let δ_t denote the set of accepted orders such that $\delta_t \subseteq \psi_t$. Acceptance of an order $q \in \delta_t$ generates undiscounted revenue $r(t, q)$ with total revenues received in a given period t defined as:

$$R(t, \delta) = \sum_{q \in \delta_t} r(t, q). \tag{3}$$

Further, define $c_q(j)$ as the capacity requirements for order q in period $t + j - 1$. Thus, the total capacity required for decision δ_t is:

$$C_{\delta_t}(j) = \sum_{q \in \delta_t} c_q(j), \quad j = 1, 2, \ldots, K. \tag{4}$$

State transitions are defined by the utilization of capacity at the start of the period plus any accepted order requirements. Three changes occur in this transition: (1) the capacity vector S_t translates in time so that the capacity utilized b_t leaves the system; (2) the amount of capacity reserved increases for any orders accepted; and (3) a new period of capacity B_{t+K} becomes available. Using our state space and capacity notation, the transitions are:

$$S_{t+1}(j) = S_t(j+1) + C_{\delta_t}(j), \quad j = 1, 2, \ldots, K-1, \tag{5}$$

$$S_{t+1}(K) = C_{\delta_t}(K). \tag{6}$$

We define this more compactly with our transition function $\tau(t, S, \delta)$.

As feasible decisions must abide by the capacity constraints ($b_t \leq B_t, \ \forall t$), this is accomplished by requiring:

$$\delta_t \in \Delta(t, S, \psi) = \{\delta \subseteq \psi : \tau(t, S, \delta) \leq [B_t, B_{t+2}, \ldots, B_{t+K-1}]\}, t = 1, 2, \ldots, T. \tag{7}$$

Define $V_t(S)$ as the optimal expected value of total discounted revenue accumulated from period t to the end of the planning horizon T given that state S is observed at the beginning of time period t. The dynamic programming recursion for DSMKP is as follows:

$$V_t(S) = \sum_{\psi \in \Psi_t} p_t(\psi) \left\{ \max_{\delta \in \Delta(t,S,\psi)} (R(t, \delta) + \alpha V_{t+1}(\tau(t, S, \delta))) \right\}, t = 1, 2, \ldots, T. \tag{8}$$

Expected revenues are discounted each period with the one-period factor α. As this is a finite horizon problem, we require the following end condition:

$$V_{T+1}(S) = 0, \forall S. \tag{9}$$

To give an idea of the size of the problem, assume we define capacity according to β discrete units in a given period. If we have a capacity window of K periods – the number of knapsacks in our problem and the number of periods which we track capacity – then there are β^K possible states in a given period. For simplicity, assume that ψ and Ψ are constant in size over the horizon. For a given arrival set ψ, there are $2^{|\psi|}$ possible accept/reject decisions to evaluate. Over T periods, this totals $T\,|\Psi|\,2^{|\psi|}\beta^K$ calculations to solve the SDP in the worst case. Note that $|\Psi|$ is exponential in the number of possible order arrivals in the set. For a moderately sized problem with $\beta = 20, K = 8$, and a maximum of five arrivals per period, this totals about 26×10^{12} calculations per period. The explosion in states and calculations, and our desire to produce a solution that can be implemented on a spreadsheet (using VBA and EXCEL), such that it could be used as a management tool, motivated the use of approximation techniques.

We have analyzed a number of different scenarios with differing numbers of orders, probabilities of arrivals, capacity requirements, and revenues. The value function, V_t, takes on a predictable shape in these instances as the state value, expectedly, is non-increasing in the amount of capacity utilized or non-decreasing in the amount of available capacity. That is, as the system fills to capacity with orders, the value of the system (through the horizon time) declines. This follows intuition as the value of an order is realized when it is accepted. However, a full system cannot accept any further orders, so its value declines as its available capacity declines. It should be clear that if one were to choose between a full and empty system, the empty system would have more potential value.

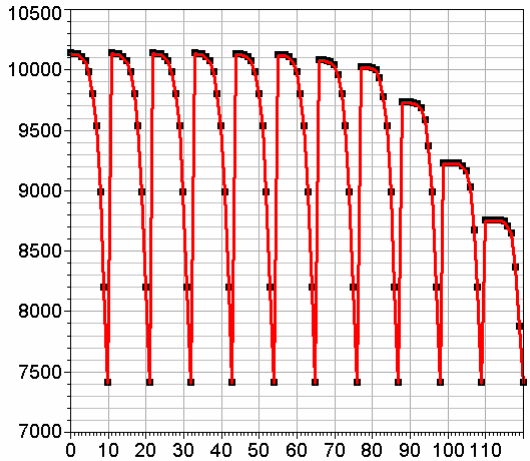


Figure 1. SDP functional equation values for various states of DSMKP.

Fig. 1 plots the values of $V_1(S)$ for a number of states S. The function is decreasing for given ranges of states, but is not monotonic due to the ordering of the states. As different components of the vector increase in capacity utilized, the value changes accordingly. If we were to order the states on a single graph (which is not possible due to the extremely large number), it would resemble Fig. 2. This is a predictable shape for diminishing value with decreasing available capacity. Knowing this shape can be quite useful when building an approximation strategy.

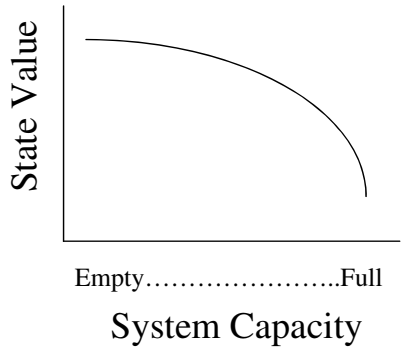


Figure 2. Decreasing SDP value function when states are ordered by available system capacity.

3. Linear programming approximation

By examining scenarios of possible arrivals for possible system states, the SDP is essentially determining optimal strategies after uncertainty is resolved over time. An approximation to this approach is to use linear programming with expected arrivals over time. That is, decisions are made on orders which are expected to arrive according to some probability. This static approach provides an approximation to the SDP.

Define x_{itj} as a binary variable to either accept or reject an order i which arrives at time t and, if accepted, stays in the system for j periods. The expected reward is r_{itj} and the expected capacity used over time is c_{itj}. Note that the expected revenues and capacities are calculated using the same probabilities, revenues, and capacities from the SDP. With the capacity set at B_t in period t we denote $\tilde{B}_t$ as available capacity because the system may not be empty at the time of solution (thus defining the state of the system). The linear program (which relaxes the binary restriction), is:

$$\max \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{j=1}^{T-t} r_{itj} x_{itj} \tag{10}$$

s.t.

$$\sum_{i=1}^{N}\sum_{k=1}^{t}\sum_{j=t-k+1}^{T-k+1} c_{ikj}x_{ikj} \quad \leq \quad \tilde{B}_t \quad \forall t = 1, 2, \ldots, T \tag{11}$$

$$0 \leq \quad x_{itj} \quad \leq 1 \quad \forall i, t, j. \tag{12}$$

The value r_{itj} is expected (and discounted) and thus incorporates the probability of order i arriving in period t and its return. All accepted orders reserve expected capacity c_{itj} in j periods. (The expected capacity, as with the expected reward, also incorporates the probability of order i arriving.) The total capacity reserved must not exceed the system capacity B_t in any period t, with $\tilde{B}_t$ representing remaining capacity after initial conditions are considered. Solving the LP over a given number of periods provides an estimate of the value function for the SDP with the given initial state.

4. Implementing the approximation

There are many approaches that can be taken when building the approximate value function. We previously experimented with simulation, but our procedures took hours as a simulation was required for each possible state in our approximation scheme (which we describe shortly). We also experimented with solving a linear program for each possible state, but due to the enormous number of possible states, this again was extremely slow. Here, we illustrate a more efficient procedure with the use of duality.

4.1. Building the approximate value function through duality

When we solve a linear program for a given state, each constraint represents the maximum capacity available in each period over time. If the capacity is fully utilized for a given period in the solution, a dual variable or shadow price is assigned to that capacity in that time period, representing the value of an additional unit of capacity at that time. According to the law of diminishing returns, this value also provides a lower bound on the last unit of capacity that was utilized.

While dual variables or shadow prices are relevant for only small changes in the right hand side of the capacity constraint, we make the assumption that the value can be interpreted over a greater span of capacity. This assumption leads us to an efficient scheme in which to approximate the value function. The idea is as follows: Solve the linear program for a given state and note its objective function value and the dual variable for each capacity constraint. The approximate value function for the initial state is merely the objective function value to the linear programming solution. The dual variables can now be used to estimate the value function for different states. For example, if we increase the amount of available capacity in period one, the value function

can be approximated by adding one unit of available capacity (one reduced unit of utilized capacity) times the appropriate dual variable to the objective function. This concept can be repeated for all possible states from the initial state to a system state of "empty." Furthermore, if we first solve the linear program for a system state of empty, that provides an upper bound such that the approximate value function is equal to the minimum of the empty system state value and that produced from the dual information. We formalize this procedure as follows, under the assumption that we can solve multiple linear programs and thus update our approximate value function accordingly:

0. Solve the linear program assuming an "empty" initial state. Define the objective function as Z_{UB} and $V_T(S)^i = 0 \; \forall S, i$.
1. For $i = 1, 2, \ldots$ Iterations Do:
2. Generate an initial state S_T^0 and solve the linear program with expected arrival information. Note the objective function value Z and shadow prices $p_1, p_2, \ldots, p_K$ for each constraint corresponding to the initial state. Set $S_T = S_T^0$.
3. Define $V_T(S)^i = Z$.
4. Do While $S_T(j) > 0, \forall j$:
5. For $j = K, K-1, \ldots, 1$ Do: $S_T(j) = \max(S_T(j) - 1, 0)$
6. $V_T(S)^i = \max(Z + \sum_{k=1}^{K} p_k \frac{S_T(k)^0 - S_T(k)}{\beta}, V_T(S)^{i-1})$
7. $V_T(S)^i = \min(V_T(S)^i, Z_{UB})$.

This approach follows our logic presented earlier in that we iterate through the states to an empty system, adding value with use of the dual variable. Note that we cannot exceed the maximum system state, defined by Z_{UB}. We also retain the maximum of any estimate for the same state in a previous iteration to maintain monotonicity.

Thus, for each linear programming solution, a new value function is generated in $O(\beta^K)$ time, where K is the number of knapsacks (reservation periods) and β is the number of units of capacity tracked in a given period (discretization).

Fig. 3 graphically illustrates our approach after iterations 1, 4, and 8. Each figure illustrates the value function approximated from a linear programming solution from a given initial state. The dual variables are then used to derive the values of the remaining, less utilized states. The approximate value function takes on our desired shape of decreasing value with increasing capacity utilization.

Fig. 4 illustrates the value function approximated with simulation and deterministic dynamic programming (generate arrivals and solve the associated deterministic dynamic program repeatedly) versus just 11 iterations of the linear programming and duality method. While the scales are different (the scale on the right hand side is for the linear program), the shape of the curve clearly

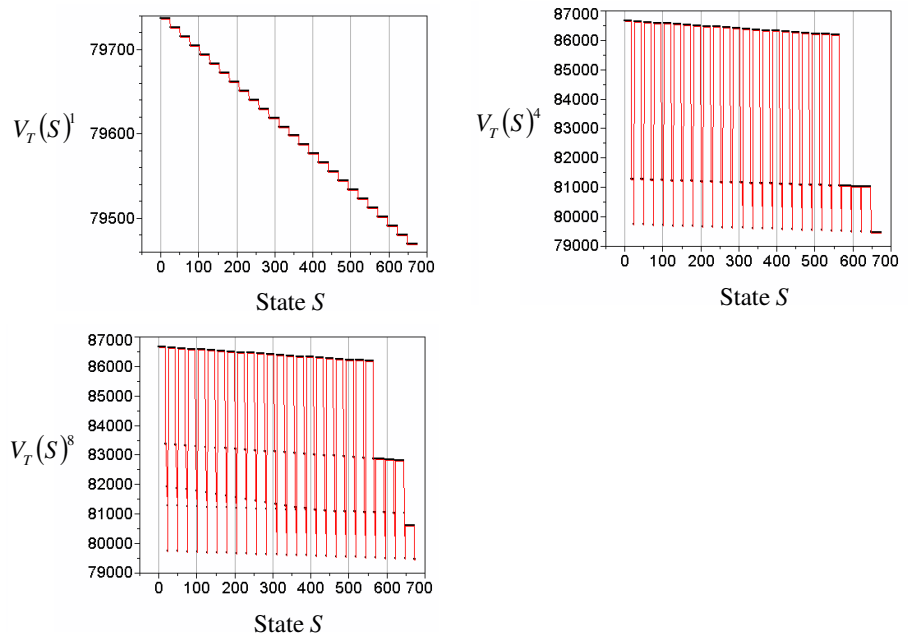


Figure 3. Iterative updates (steps 1, 4, and 8) of approximate value function using linear program solutions (objective function value and dual variables).

follows what we want. Needless to say, the linear programming approximation was generated in minutes while the simulation approximation took hours.

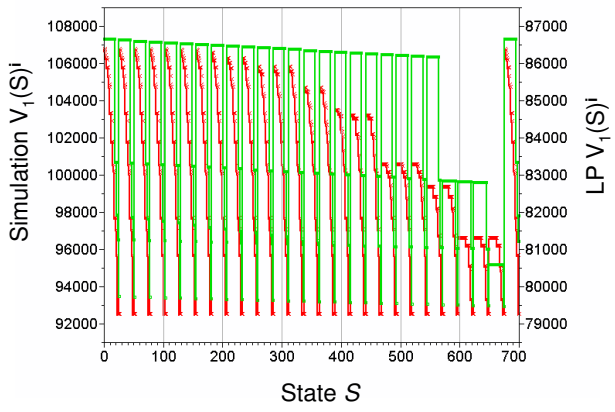
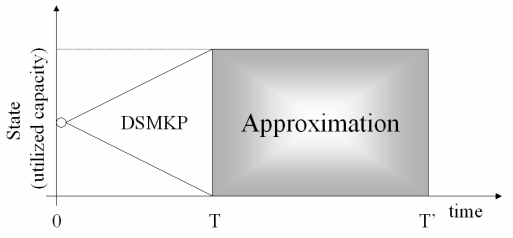


Figure 4. Approximate value functions generated with simulation (circle points) and linear programming with duality (square points).

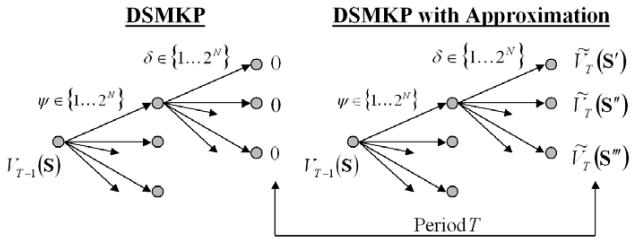
Note that we are not overly concerned whether the linear program over (or under) estimates the value function. Rather, we are more interested in appropriately *differentiating* the value of states so that the dynamic program more accurately chooses between options (future possible states). Thus, we are more interested in estimating the relative difference between states, not their absolute values. Empirically, the linear programs and estimates derived with dual variables, accomplish this goal.

4.2. Integrating the approximate value function into SDP

Once we have an estimate of the value function, it must be integrated into our SDP. Figs. 5(a) and (b) illustrate our approximation approach.



(a)



(b)

Figure 5. Approximation scheme (a) solved over periods T through T' in order to (b) provide non-zero state values at time T for the SDP.

Fig. 5(a) illustrates the overall scheme in that a valid network of states can be built over T periods. This network, beginning with a single node at time one which represents the initial state, only generates states that can be reached from the initial node. Whereas the boundary conditions at time T for a typical SDP implementation would assume state values of zero, we solve the approximation from periods T through T' and, as noted in (b), populate the SDP valid states in period T with these values. Fig. 5(b) illustrates one segment of a network

describing the SDP, with a node representing a state and arcs representing possible arrivals and decisions. With N possible order arrivals, there are 2^N possible arrival combinations, leading to a maximum of 2^N decisions for a given arrival combination.

Thus, we can solve linear programs and generate the value function. Once complete, this data is fed to the valid network which is solved backwards using the SDP recursion. It is hoped that substituting the approximate values provides a better differentiation of states than the assumed boundary condition of zero for each state. We empirically test this in the following section.

5. Large scale implementation

We chose a large scale application, with up to nine possible arrivals in a period (2^9 possible combinations), to illustrate the approximation method. The data was generated to mimic a typical manufacturing environment in that low margin, high volume jobs had a relatively higher probability than high margin, low volume jobs. The arrivals and their expected rewards, arrival probabilities, and capacity requirements are defined in Table 1.

Thus, the set Ψ consists of the 2^9 possible combinations and an arrival realization ψ comes from this set. As the arrivals are independent, the probability of ψ arriving, $p_t(\psi)$, is calculated by multiplying the arrival probabilities (from Table 1) of each individual order in the set by the probability of the remaining orders not arriving (one minus the value given in Table 1).

Table 1. Expected information for nine classes of arrivals over time.

Arrival	Volume Class	Margin Class	Probability of Arrival	Margin	Reward	Required Capacity	Life
1	Low	Low	5%	30%	$857	10%	4
2	Medium	Low	10%	30%	$1714	20%	4
3	High	Low	75%	30%	$2571	30%	4
4	Low	Medium	20%	50%	$2000	10%	4
5	Medium	Medium	50%	50%	$4000	20%	4
6	High	Medium	35%	50%	$6000	30%	4
7	Low	High	25%	70%	$4666	10%	4
8	Medium	High	7%	70%	$9333	20%	4
9	High	High	2%	70%	$14,000	30%	4

We discretized capacity according to 26 buckets (0 to 25) in a period, or 4 percent increments. We further assumed that capacity could be reserved four periods out into the future (i.e. four knapsacks in our approach). This might correspond to quarters with capacity reservations available for one year. A 2.50% quarterly interest rate was used for discounting.

Utilizing Visual Basic for Applications (VBA) and Microsoft Excel on an IBM-compatible personal computer with a Pentium 3, 1GHz processor and 768MB RAM, we were able to solve five periods of the SDP for each problem instance (specific run times are in Table 4), although a number of instances took excessive time (over 15 hours) to solve.

We generated nine arrivals at the first period from the expected set of arrivals and solved the problem with the SDP (values of zero assigned to states in the final time period) and the SDP with the linear programming approximation scheme described in this paper. The nine arrivals at time one are given in Table 2. (These were generated from our expected data set in Table 1 under the assumption that the rewards, capacity, and lifetimes were uniformly distributed.) Table 2 also defines the nine instances examined in this study, each with a different initial state of utilized capacity. For example, an initial state of $[5, 5, 5, 5]$ means that 5/25, or one-fifth of capacity is already reserved in each of the next four periods.

Table 2. Initial states of nine instances studied with nine actual (generated) arrivals at time one for accept/reject consideration.

Instance	Initial State	Arrival	Reward	Capacity	Life
1	$[0, 0, 0, 0]$	1	\$2748	32%	4
2	$[5, 5, 5, 5]$	2	\$4000	23%	3
3	$[11, 11, 11, 11]$	3	\$4666	11%	3
4	$[17, 17, 17, 17]$	4	\$5988	30%	4
5	$[23, 23, 23, 23]$	5	\$957	14%	2
6	$[23, 17, 11, 5]$	6	\$931	13%	1
7	$[5, 11, 17, 23]$	7	\$1111	13%	1
8	$[0, 1, 2, 3]$	8	\$961	8%	2
9	$[3, 2, 1, 0]$	9	\$994	10%	2

For the approximation, we solved 11 linear programs over a 19-period horizon and utilized the objective function values and dual variables to generate the approximate value function. It took approximately 2 seconds to solve each linear program and acquire the dual variables (using Solver in Excel). The procedure to generate the approximate state values took 113.14 seconds. Thus, the entire approximation procedure took about 135 seconds.

Table 3 summarizes the first decisions for each of the solution approaches over two, three, four, and five periods. The table has nine digits with a one representing accepting that arrival and a zero representing a reject, such that "011001111" means that arrivals 1, 4, and 5 should be rejected but the remaining (2,3,6,7,8,9) should be accepted. (These nine arrivals correspond to those in Table 2.) The base 10 equivalents of these binary decisions are also given in the table in order to make comparisons more easily.

Table 3. Optimal first decision for nine arrivals and given initial state with and without approximation.

	SDP				SDP with Approximation			
Instance	Pds: 2	3	4	5	2	3	4	5
1	111100000	011101100	001101111	011001111	011101100	001101111	011001111	011001111
	480	236	111	**207**	236	111	**207**	207
2	011100001	001101100	011001101	011001101	001101100	001001111	011001101	011001101
	225	108	**205**	205	108	79	**205**	205
3	001100001	001001101	001001101	001001101	001001100	001001101	001001101	001001101
	97	**77**	77	77	76	**77**	77	77
4	000100000	001000100	001000100	001000100	001000100	001000100	001000100	001000100
	32	**68**	68	68	**68**	68	68	68
5	000000000	000000000	000000000	000000000	000000000	000000000	000000000	000000000
	0	0	0	0	**0**	0	0	0
6	000100000	000100000	000100000	001000100	001000100	001000100	001000100	001000100
	32	**68**	68	68	**68**	68	68	68
7	000001111	000001111	000001111	000001111	000001111	000001111	000001111	000001111
	15	15	15	15	**15**	15	15	15
8	011100101	001101101	011001111	011001111	001101100	001011111	011001111	011001111
	229	109	**207**	207	108	95	**207**	207
9	011100011	001101100	011001111	011001111	001101100	011001111	011001111	011001111
	227	108	**207**	207	108	**207**	207	207

Examining Table 3, one sees that five of the nine large-scale problems arrive at the same initial decision in the same horizon. For example, both solution approaches arrive at the decision of 205 for instance 2 (an initial state of $[5, 5, 5, 5]$) when solving the SDP over four periods. (The decisions given in bold in Table 3 are believed to be the optimal time zero decisions for each instance.) This agreement on the solution in the same time frame occurs in instances 2, 3, 5, 7, and 8.

We see the value of the approximation for the other four instances, 1, 4, 6, and 9, as the approximate procedure arrives at the time zero decision one time period earlier than the SDP. (We can only speculate that this is the optimal time zero decision for an infinite horizon problem, but it is clear that both methods agree on the decision and the approximation method discovered it earlier.) While this is not rousing evidence to support the use of approximation, it is clear that it helps in terms of requiring fewer periods of analysis.

This becomes more important when one examines the run times in Table 4. This is the time it takes to build the reachable network and solve the SDP recursion over the reachable states. When one notices that it takes over 21,000 seconds (nearly 6 hours) to solve instances 1 and 9 over four periods and almost 62,000 seconds (over 17 hours) to solve instance 1 over five periods, the benefit of the approximation becomes more pronounced. An average of over 9000 seconds (2.5 hours) per instance is saved through the approximation procedure as

Table 4. SDP solution times (in seconds). This includes building the network of reachable states and solving the recursion.

Instance	State\Periods :	2	3	4	5
1	$[0, 0, 0, 0]$	82	6807	26,140	61,920
2	$[5, 5, 5, 5]$	39	2048	5600	26,900
3	$[11, 11, 11, 11]$	15	566	1052	7767
4	$[17, 17, 17, 17]$	21	154	317	1069
5	$[23, 23, 23, 23]$	8	16	15	42
6	$[23, 17, 11, 5]$	9	170	1058	12,447
7	$[5, 11, 17, 23]$	9	125	100	179
8	$[0, 1, 2, 3]$	68	4132	11,978	39,060
9	$[3, 2, 1, 0]$	61	4885	21,506	58,020
	Average:	35	2100	7530	23,045

solutions are found one period sooner in four instances. This is valuable time that the decision-maker can spend improving estimates of future arrivals.

There is further evidence which supports the use of the approximation when examining the solutions. It should be clear that arrival class 7 is quite desirable as it is defined by a high margin and low volume. Surprisingly, it does not appear in the SDP solution (without approximation) for the 2-period solution in capacity cases 1, 2, and 9 and does not appear in all feasible solutions until the 3-period problem is solved. However, these orders are accepted in the approximation solution for the 2-period solution in all feasible cases.

This analysis also illustrates that there may be value in solving more linear programs and updating the value function estimates accordingly. Recall that we only solved 11 linear programs. This decision was made based on graphical evidence that the value function estimates were not changing drastically in the last few iterations. However, different linear programming solutions and dual variables may lead to better differentiation between states and provide more evidence for use of the approximation.

6. Conclusions and directions for future research

We have presented a method in which we approximate the value function of a stochastic dynamic programming approach for a dynamic, stochastic multiple knapsack problem. The model has been used in an order accept/reject setting for manufacturing applications. Unfortunately, the model grows exponentially in the number of orders and the discretization of the state space.

Our approximation scheme was based on solving a linear programming approximation to the DSMKP. For a given initial state of the system, the objective function value and dual variables associated with the value of additional capac-

ity were used to build an approximate value function quite efficiently. These values were then used as substitutes for the boundary conditions of zero when solving the SDP over a given horizon.

The benefits of this approach are that the value function can be constructed relatively quickly (in just over two minutes in our large-scale application) and the values provided some measure of valid differentiation between states for the SDP in the final period. This is clear when examining the different solutions found when solving a large-scale application over a number of different time periods. The true benefit of the approximation is that optimal initial solutions may be found sooner, by solving the SDP over shorter horizons, because the approximate values help reduce the end-of-study effects induced by assuming boundary condition values of zero. This is turn helps to reduce the computational burden of the SDP, as a smaller horizon problem can be solved. This is critical when one is constrained to solving problems in a timely manner and on a desktop computer.

Future research may examine how one can more formally determine the appropriate number of linear programs to solve when generating the approximate value function. We employed a simple graphical method. Furthermore, we may examine different ways of integrating linear programming solutions with dual variables. For example, we utilized the dual variables for all states between the initial state assumed when solving the linear program and the empty system state. However, we could assume the values are only to be used in a specified range and thus only update smaller ranges with dual variables. This should improve the estimate of the value function at the cost of having to solve more linear programs.

It should also be clear that other functions could be used to approximate the end-of-horizon values. We employed linear programming, but one could also directly approximate the value function itself through sampling and interpolation methods. The benefit of this approximation scheme is that any function can be utilized in the first phase and the importance of the approximation is to provide a differentiation between states at the horizon.

7. Acknowledgments

We would like to thank two referees whose comments greatly improved the readability of this manuscript. We would also like to recognize National Science Foundation funding for this research through grants DMI-9984891 and DMI-0121395.

References

Bellman, R.E. (1957) *Dynamic Programming.* Princeton University Press, Princeton, NJ.

Bertsekas, D.P. and Tsitsiklis, J. (1996) *Neuro-Dynamic Programming.* Athena Scientific.

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

de Farias, D.P. and Van Roy, B. (2003) The linear programming approach to approximate dynamic programming. *Operations Research* **51**, 850-865.

Godfrey, G. and Powell, W.B. (2002) An adaptive, distribution-free algorithm for the newsvendor problem with censored demands, with application to inventory and distribution problems. *Management Science* **47**(8), 1101-1112.

Johnson, S.A., Shoemaker, C.A., Stedinger, J.R., Li, Y. and Tejada-Guibert, J.A. (1993)Numerical solutions of continuous-state dynamic progams using linear and spline interpolation. *Operations Research* **41**, 484-500.

Perry, T.C. and Hartman, J.C. (2004) Allocating manufacturing capacity by solving a dynamic, stochastic, multiple knapsack problem. Technical Report T004-009, Industrial and Systems Engineering, Lehigh University, Bethlehem, PA.

Philbrick, R. and Kitanidis, P.K. (2001) Improved dynamic programming methods for optimal control of lumped parameter stochastic systems. *Operations Research* **49**, 398-412.

Powell, W.B. and Carvalho, T. (1998) Dynamic control of logistics queueing networks for large scale fle et management. *Transportation Science* **32**(2), 90-109.

Shapiro, J., Powell, W.B. and Simao, H.P. (2002) An adaptive, dynamic programming algorithm for the heterogeneous resource allocation problem. *Transportation Science* **36**(2), 231-249.

Topaloglu, H. and Powell, W.B. (2006) Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems. *INFORMS Journal on Computing.*