INDUSTRIAL ENGINEERING APPLICATIONS AND PRACTICE: USERS ENCYCLOPEDIA®

DYNAMIC PROGRAMMING

Problem Statement

Dynamic Programming is an approach developed to solve sequential, or multi-stage, decision problems; hence, the name "dynamic" programming. But, as we shall see, this approach is equally applicable for decision problems where sequential property is induced solely for computational convenience.

Unlike other branches of mathematical programming, one cannot talk about *an* algorithm that can solve *all* dynamic programming problems. For example, George Dantzig's Simplex Method can solve *all* linear programming problems. Dynamic programming, like branch and bound approach, is a way of *decomposing* certain hard to solve problems into equivalent formats that are more amenable to solution. Basically, what dynamic programming approach does is that it solves a multi-variable problem by solving a series of single variable problems. This is achieved by tandem projection onto the space of each of the variables. In other words, we project first onto a subset of the variables, then onto a subset of these, and so on.

The essence of dynamic programming is Richard Bellman's ***Principle of Optimality***. This principle, even without rigorously defining the terms, is intuitive:

> *An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

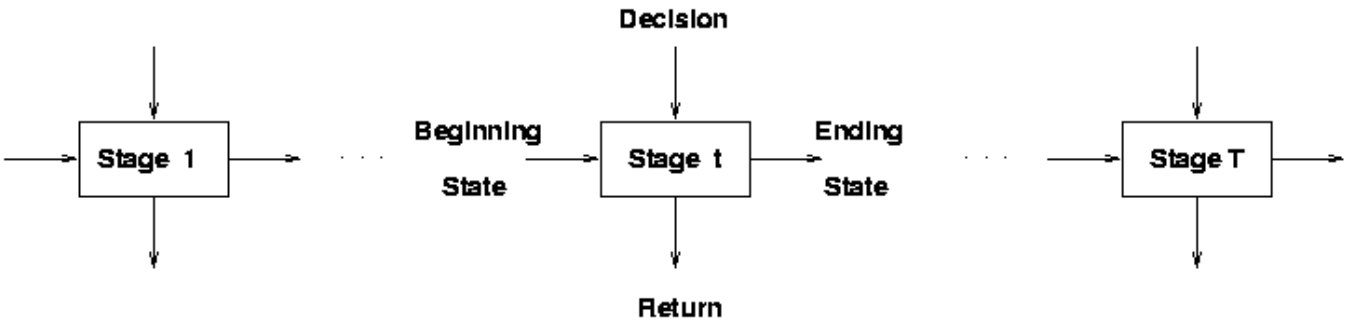
This is a self-evident principle in the sense that a proof by contradiction is immediate. Rutherford Aris restates the principle in more colloquial terms:

> *If you don't do the best with what you have happened to have got, you will never do the best with what you should have had.*

In this brief introduction, *deterministic* dynamic programming approach will be illustrated based on two simple examples. There are excellent books on various aspects of dynamic programming, but probably the best place to start for further study of this topic is the introductory operations research books by Hillier & Lieberman or Taha. Also Moshe Sniedovich's page provides further introductory material on dynamic programming.

Let us first discuss a "dynamic" problem which is composed of time dependent stages, referred to as *periods*. Later, we shall discuss an example in which sequential decision making is not in the nature of the problem but induced for computational reasons.

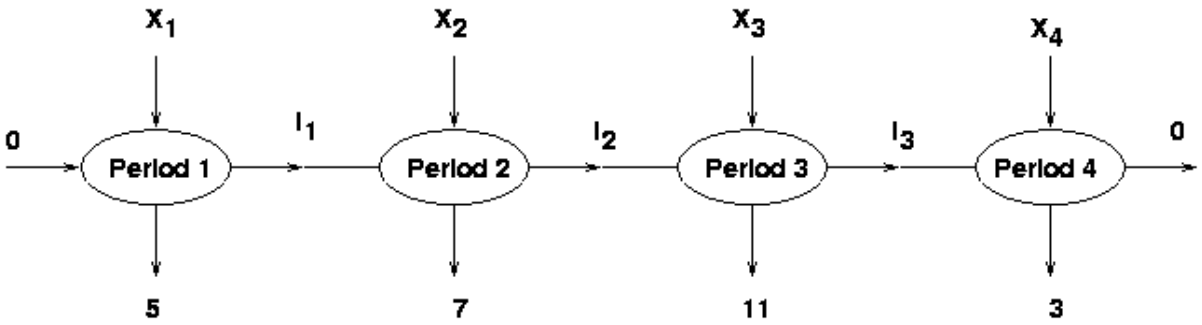
Consider the following schematic representation:



The state variable at a particular stage has to be defined in such a way that it completely describes the process. Given the state of the process at the beginning of a stage, we make a decision which results in a specific return and and transforms the process to the ending state at the end of the stage. The objective is to maximize (or minimize, depending on how the return is defined) total returns over all stages. Let us clarify these concepts in the context of an example: the fixed-cost, periodic review inventory process with dynamic demand --the so-called Wagner-Whitin Model. Suppose you are faced with the demands, $D = \{5, 7, 11, 3\}$, for the next four periods. When you decide to produce at any period, t, you incur a fixed charge (setup) cost, k_t, and a production cost of c_t per unit.

Period, t	Setup Cost, k_t	Unit Production Cost, c_t
1	5	1
2	7	1
3	9	2
4	7	2

Furthermore, for every unit carried in inventory from one period to the next, you incur an inventory holding cost of $1 per unit. Shortages are not allowed and you do not have any inventory at the start of the Period 1. Letting X_t denote the amount produced in Period t and I_t denote the amount of inventory we have at the end of Period t, we can depict the problem as in the following flow network:



At any stage, the beginning inventory plus the amount produced during a period must be equal to the demand plus ending inventory at the end of the period. In terms of dynamic programming formulation, the "state" of this inventory process is completely described by the level of beginning inventory at each period. In other words, in order to make optimal decisions for the current and all future periods all we need to know is the current inventory level. How we ended up with that amount

is immaterial to our current and future decisions. This is known as the *Markovian Property*. For a process to be *Markovian*, the future must depend only at the present and past should not have any effect on the future. In order for Bellman's *Principle of Optimality* to be applicable, thus to have a dynamic programming formulation of the decision problem, the underlying process must be *Markovian*.

The inventory process described in the above example is clearly *Markovian*: in order to make the current and future procurement decisions all I need to know is the amount I have on hand. On the other hand, if perishability has to be taken into account, in addition to the total amount on hand, I need to know when each item in the inventory was produced. In this case, the past matters and the inventory process, with the current definition of the state variable, is no longer *Markovian*. But modifying the definition of the state variable, the process can be made *Markovian*. Suppose I define the state of the process not by a single number representing the current inventory level, but by a "matrix" with two columns. First entry in each row represents the "age" of an item, and the second entry represents the amount we have on hand of the items with that age. By carrying all the past information in the current state vector, we are forcing the process to be *Markovian*. This, though, is achieved at a cost of enlarging state space which causes considerable computational burden, and referred to as *curse of dimensionality* by Richard Bellman.

Before we discuss how dynamic programming solves the above multi-stage problem, let us look into another problem in which the sequential nature is not apparent. Consider the following integer programming problem taken from Taha:

$$\text{MAX } 8x_1 + 7x_2,$$

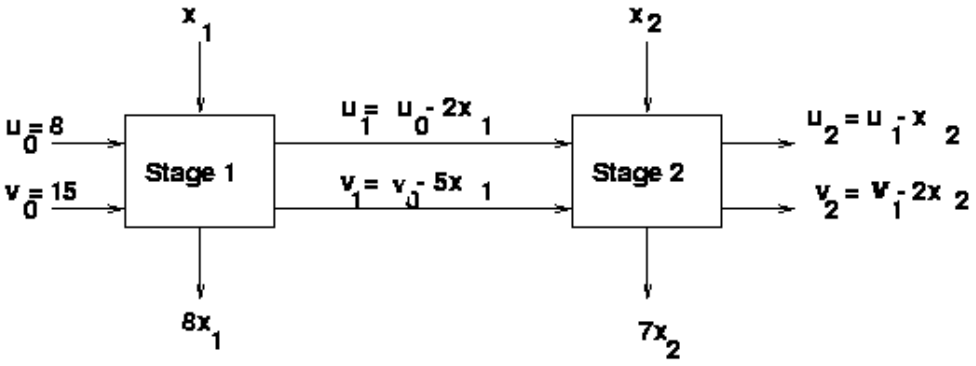
SUBJECT TO:

$$2x_1 + x_2 <= 8,$$

$$5x_1 + 2x_2 <= 15,$$

$$x_1, x_2 >= 0, \textit{ and integer.}$$

If we interpret x_1 as the value of activity 1 and x_2 as the value of the activity 2, then we can define each stage as the process of deciding on the level of each activity. One can interpret the right hand side values of the constraints as the resource availabilities. At each stage we decide on the level of an activity, thus consuming certain amount from each of the resources. This decision is constrained by the available amounts of each resource at the beginning of the stage, and unused amounts of the resources are left to be used by the activities in the "downstream" stages. Thus, knowing "available amounts of each of the resources at the beginning of a stage" is sufficient to make optimal decisions at the current and all of the remaining stages of the process. Since we have two constraints in the above problem, we need to define state of the process by a vector of two elements. Let $u_{(t-1)}$ be the available amount of first resource and $v_{(t-1)}$ be the available amount of second resource just before we make the decision x_t. Schematically, this multi-stage decision process can be shown as:

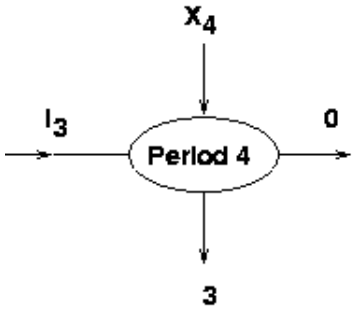


There is no inherent sequential character in this integer programming problem: I could have equally defined the first stage as the stage at which decision is made concerning the value of x_2, and the second stage for x_1.

Solution to the Problem

As mentioned earlier, the main advantage of using dynamic programming is instead of solving one large problem, we solve many small problems. Let us get back to the dynamic lot sizing problem introduced earlier. It can be shown that in this problem it will be optimal to produce only at periods where beginning inventory is zero. That is, we either have a beginning inventory or we produce at a stage, but not both. The idea behind this reasoning is the following: since there is no capacity limit on production, if it is profitable to carry any amount of inventory into a period, we should carry at least the total demand for that period. It will not make sense to meet part of the demand from inventory and the rest from the production in that period. If production will take place in a period, then it should not make sense to have items carried in inventory into that period. This result implies that if there will be production in any period, the amount produced is either equal to the exact demand of that period, or equal to the demand of that period plus the demand of the next period, or to the next three periods, and so on.

Suppose we are at the beginning of last stage, Period 4:



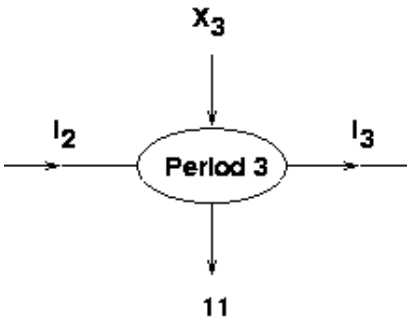
What are all the possible values of the state vector, I_3? In other words, what are the possible beginning inventory levels at the beginning of Period 4? It is either 0, in this case we need to produce exactly 3 units to meet the demand; or the beginning inventory is 3 units, and we do not need to produce any amount. Any other value for the beginning inventory level will not be optimal in the light of the above property of this type of problems. Thus the state space for this stage consists of two elements, {0, 3}.

The dynamic programming approach to this problem then asks, what will be the production level if $I_3 = 0$, and what if $I_3 = 3$? Being the last stage, there are no alternatives; that is, there is nothing to optimize. If $I_3 = 0$, then $X_3 = 3$, with a total

cost of \$13 [= \$7 (for the setup) plus \$6 (3 units at \$2 each)], and if $I_3 = 3$, then $X_3 = 0$, with a total cost of \$0 [since there is no production, and ending inventory is zero). We shall keep these numbers in mind, as in the following table,

I_3	X_4	COST
0	3	13
3	0	0

and go back one stage to the beginning of Period 3:



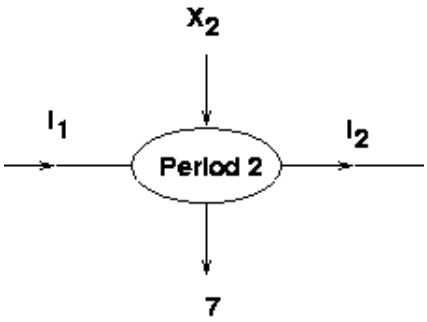
There are three alternative values for the beginning inventory level, I_2, at this stage: 0, 11, or 14 [= 11 + 3]. If $I_2 = 0$, then $X_3 = 11$ or 14. When $X_3 = 11$, after meeting the demand for Period 3, the remaining inventory $I_3 = 0$; and, when $X_3 = 14$, I_3 becomes 3. If there is a beginning inventory, i.e when $I_2 = 11$ or 14, then there cannot be any production in Period 3, and after meeting the demand for this period, the ending inventory, I_3, is 0 and 3, respectively. Now we are ready to compute the costs incurred in this plus all the future states for each of the production decision alternatives. For example, if $I_2 = 0$, and we decide to produce 14 units in Period 3, we will end up with an ending inventory of 3 units. Costs in this period will be: \$9 for setup, plus \$2 for each of the 14 units, \$28, plus the cost of carrying three units of ending inventory into the fourth period at \$1 each, \$3; which add up to a total of \$40. The beginning inventory for Period 4, I_3, will be 3 units and we recall (or, read off from the previous table) that corresponding cost in that and all future stages will be zero. Thus, the total cost for this alternative is \$40. This and all other alternatives are shown in the following table for Stage 3:

I_2	X_3	I_3	COST
0	11	0	44
	14	3	40
11	0	0	13
14	0	3	3

If $I_2 = 11$, or 14, there is nothing to decide; in either case production level will be zero, $X_3 = 0$. But when $I_2 = 0$, then we can produce either 11 units or 14 units in Period 3. Since it is more economical to produce 14 units (\$40 versus \$44), we cross out the alternative $X_3 = 11$, and have the final table for Period 3:

I_2	X_3	I_3	COST
0	14	3	40
11	0	0	13
14	0	3	3

Keeping the above figures in mind, we proceed backwards to Stage 2:



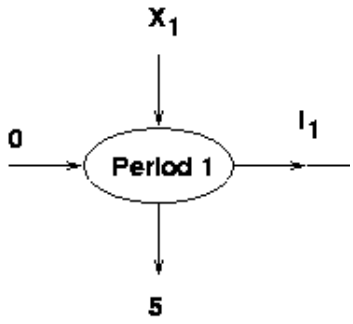
Now the state space for this stage consists of the numbers {0, 7, 18, 21}. Again decision alternatives exists only when $I_1 = 0$, with the associated costs as shown in the following table:

I_1	X_2	I_2	COST
0	7	0	54
	18	11	49
	21	14	45
7	0	0	40
18	0	11	24
21	0	14	17

After eliminating the more costly alternatives we end up with the final table for Stage 2:

I_1	X_2	I_2	COST
0	21	14	45
7	0	0	40
18	0	11	24
21	0	14	17

Finally, at the beginning of Stage 1,



the state space consists of a single number, $I_0 = 0$, and we have four alternatives for production level in this stage, $X_1 = 5$, 12, 23, or 26. Corresponding costs are given in the following table:

I_0	X_1	I_1	COST
0	5	0	55
	12	7	64
	23	7	70
	26	21	69

Thus it is optimal to produce 5 units in Period 1, resulting with an ending inventory of 0 units. Tracing the solution back to the previous table, with $I_1 = 0$, it is optimal to produce 21 units in Period 2, which will be sufficient to meet the demand for all the remaining stages. The optimal policy, then, is $X^* = \{5, 21, 0, 0\}$ with a total cost of $55.

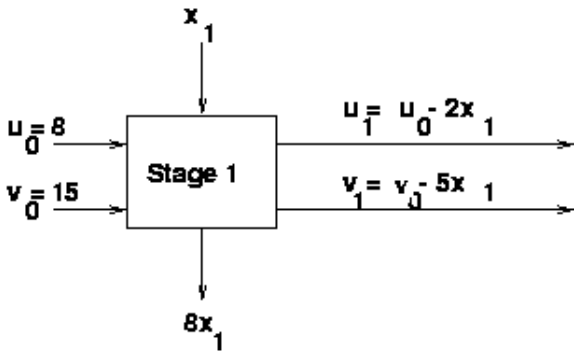
In the above problem we have proceeded backward starting at the last stage. This is referred to as *backward recursion* in dynamic programming. In some problems, as is the case in the following example, the reverse of this, *forward recursion*, seems to be more convenient. Clearly there are more efficient ways of solving integer programming problems, but purely for illustrative purposes, let us look into dynamic programming approach for solving the following mathematical program:

$$Max\ 8\,x_1 + 7\,x_2$$

$$2\,x_1 + x_2 <= 8$$
$$5\,x_1 + 2\,x_2 <= 15$$

$$x_1, x_2 >= 0, \text{ and integer}$$

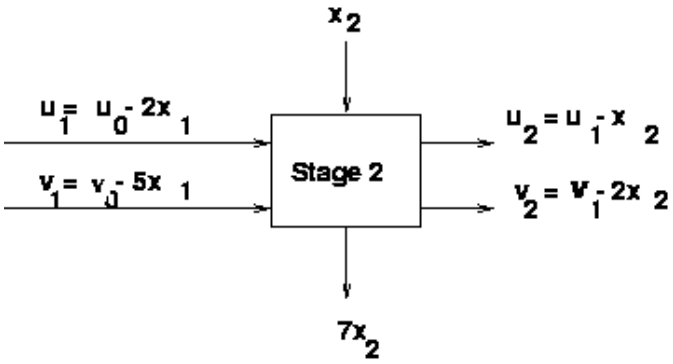
The multi-stage formulation of this problem and the definition of state variables were discussed in the previous section. Now we start with the first stage:



Initial state vector at the beginning of Stage 1 is (8, 15). Observing this, we need to assign a value for x_1. The only constraint we have is that the state vector at the end of this stage is to be non-negative, i.e. $u_1 = 8 - 2x_1 >= 0$ and $v_1 = 15 - 5x_1 >= 0$. Thus, the possible values x_1 can assume are 0, 1, 2, and 3. These values and the resulting *return* (i.e. the contribution to the objective function value, i.e. $8x_1$) from this stage (and all the "previous" stages; but this being the first stage, there are no previous stages) are summarized in the following table:

(u_0, v_0)	x_1	(u_1, v_1)	**RETURN**
(8, 15)	0	(8, 15)	0
	1	(6, 10)	8
	2	(4, 5)	16
	3	(2, 0)	24

Now, we proceed *forward* to Stage 2:



For each value of the state vector at the beginning of Stage 2, the maximum value that can be assigned to x_2, such that the ending state vector to be non-negative, is given in the following table:

(u_1, v_1)	x_2	RETURN
(8, 15)	7	49
(6, 10)	5	43
(4, 5)	2	30
(2, 0)	0	24

The optimal solution is $x^* = (0, 7)$ with an objective function value of 49.

The above two examples are simple illustrations of how dynamic programming can be used in solving certain optimization problems. As the problems become more involved, one has to be more rigorous in defining how one goes from one stage to the next. This necessitates constructing *functional equations* that facilitate these manipulations. For a detailed discussion, refer to the specialized texts such as the books by Bertsekas and Sniedovich.

Throughout this discussion we were working on *deterministic* problems; given the initial stage and the decision we make at a certain stage, the resulting state of the systems were known with certainty. These type of problems are amenable to *deterministic* dynamic programming. When the resulting state is not known with certainty but its realization is governed by a probability distribution, we have a *stochastic* dynamic programming problem. A thorough discussion of *stochastic* dynamic programming problems can be found in the above mentioned books.

Ömer S. Benli
California State University, Long Beach