DYNAMIC PROGRAMMING

Sniedovich M.
The University of Melbourne, Parkville, Australia

Keywords: Bellman, curse of dimensionality, decomposition, dynamic programming, functional equation, invariant embedding, Markovian condition, monotonicity condition, objective function, optimization, policy, policy iteration, principle of optimality, recovery procedure, sequential decision process, stage, state, stationary, stochastic processes, successive approximation, transition function, value iteration

Contents

Summary

One of the most common approaches to dealing with complex problems is *decomposition*. The basic idea behind this approach is simple and intuitive: a complex problem is decomposed into sub-problems and the solution to the complex problem is constructed from the solutions found for the sub-problems. Dynamic programming (DP) is based on this simple idea, except that it applies it (repeatedly) to the sub-problems as well. For historical reasons, the conceptual framework of DP is that of an *optimization*

problem cast as a *sequential decision problem*. That is, in its traditional role DP deals with situations where a sequence of decisions is to be made so as to optimize a given function of the decisions variables subject to some constraints imposed on these variables. In this framework decomposition is achieved by fixing a decision and considering the (sub) problem associated with the remaining decisions. This leads to a *functional equation* stipulating the relationship between the optimal values attained by the objective functions of the sub-problems. The optimal solution to the original problem is obtained by solving this functional equation. Needless to say, for this strategy to make sense and provide an optimal solution to the original problem, the sub-problems generated by the decomposition scheme must be consistent with the original problem. This consistency property is captured by what Richard Bellman—the Father DP—called *the principle of optimality*, which over the years has become synonymous with DP.

1. Introduction

In 1952 a young mathematician by the name of Richard Bellman published a very short article—in fact 4 pages long—entitled *On the Theory of Dynamic Programming*. In it Bellman sketched the foundation of a new approach to problem solving—widely known today as Dynamic Programming (DP). This approach offers practical problem solving tools in many diverse application areas such as network optimization (see *Graph and Network Optimization*), project management, decision analysis (see *Decision Trees and Influence Diagrams*), reservoir control, inventory problems, artificial intelligence, computer science, agriculture, forestry, finance, biology, cutting stock, quality control, reliability, publishing and text processing, resource allocation, medicine, military and recreation. This list is not complete.

Although numerous books and articles have been written on DP, the question "What is DP?" is as relevant today as it was 50 years ago, perhaps even more so. For as Bellman warned at the outset, although DP is based on extremely basic principles, it is difficult to capture its essence in a straightforward, rigid, mathematical formalism. Indeed, it is the opinion of many experts, that a number of important aspects of DP are more art than science. One of the manifestations of this fact is the popular "learn/teach DP by example" strategy with regard to DP teaching and learning. This is not to say that it is difficult to compose a comprehensive, rigorous, formal mathematically oriented formulation for DP. In fact, this has been done numerous times since the mid-1950s. What is difficult is to accomplish this and yet keep the formulation simple, easy to grasp and useful.

As indicated above, Bellman was fully aware of this fact and his first book on DP—dated 1957—was written with this issue in mind. This, needless to say, attracted a fair bit of criticism from scholars who argued that Bellman's formulation of DP was not rigorous. It is not surprising therefore that attempts have been made in the 1960s and 1970s to develop more rigorous mathematical formulations for DP.

This early chapter in the history of DP is vividly reflected in the present state of the art and thus naturally has its impact on the present discussion. However, it will be constructive to begin our guided tour with a rather abstract description of DP:

DP is an approach to *problem solving* utilizing the following meta strategy:

Step 1: Generalize your problem by transforming it into a *family* of related problems using one or more features of the problem as *parameters*.
Step 2: Derive *a functional equation* relating the solutions to each of these problems to the solutions of the others.
Step 3: *Solve* the functional equation.
Step 4: Use the solution to the functional equation to *recover* a solution to your particular problem.

As clearly demonstrated by several generations of students all over the world, this is easier said than done, hence the notion "the art of DP".

Before we translate the above recipe into something more concrete, it will be instructive to apply it to four seemingly disparate but representative problems.

2. Preliminary Examples

The objective of the four examples examined in this section is two-fold, namely to illustrate the above recipe in action and to sketch the general profiles of problems that DP is usually applied to. With regard to the latter, note that the first example is *a computational problem*, the second is a *proof problem*, the third is a *decision problem* and the fourth is an *optimization problem*. Each problem is defined by a short description of a given object and a task to be performed in relation to it.

We have purposely chosen the first two problems to be very simple and well known so as to emphasize that DP is very pervasive and that it is used daily by numerous persons most of which are not aware of the fact that they use DP.

Example 1:

Given: A list of numbers, $\boldsymbol{x} = (x_1, ..., x_m)$.

Task: Compute the sum of the elements of the list.

Step 1: The task is to compute the value of $x_1 + \cdots + x_m$. Thus, a simple way to generalize the given problem and create out of it a family of related problems is to consider the task of computing the *partial sums* of *x*. That is, let

$$Sum(n) := x_1 + \cdots x_n,\ n = 1, 2, \ldots, m \qquad (1)$$

so that now we have to solve *m* problems.

Step 2: From the definition of *Sum(n)* it follows that

$$Sum(n+1) = Sum(n) + x_{n+1},\ n = 1, 2, ..., m-1. \qquad (2)$$

This is a typical DP functional equation.

Step 3: It is very easy to solve the functional equation: set $Sum(1) = x_1$ and then compute the right-hand side of (2) for $n = 2,3,\ldots,m-1$—in this order.

Step 4: The value of interest is Sum(m) and it is obtained as the last item computed in Step 3 of the above procedure. ♦

Example 2:

Given: An arithmetic series $x_0, x_1, x_2, \ldots$ such that $x_0 = a$ and $x_{j+1} = x_j + d$ for $j = 0,1,2,3,\ldots$ where *a* and *d* are given numbers.

Task: Prove that $x_{37} = x_0 + 37d$.

Step 1: We generalize the given problem by rephrasing it as follows: Prove that

$$x_j = a + jd,\ j = 0,1,2,3,\ldots \qquad (3)$$

Step 2: In this example we are also given the functional equation relating members of the family of problems under consideration, namely:

$$x_{j+1} = x_j + d,\ j = 0,1,2,3,\ldots. \qquad (4)$$

Step 3: The proof consists of solving the functional Eq. (4) (*by induction* on *j)* showing that the solution satisfies (3). For $j = 0$ the functional Eq. (4) yields $x_1 = x_0 + d = a + d$, so we conclude that the inductive hypothesis (3) is clearly valid for $j = 0$. We thus assume that the inductive hypothesis is valid for $j = 1,2,3,\ldots,m$, for some $m > 0$, and consider $j = m+1$. The functional equation asserts that $x_{m+1} = x_m + d$, so utilizing the inductive hypothesis for $j = m$, namely $x_m = a + md$ we obtain $x_{m+1} = a + md + d\,x = a + (m+1)d$. Hence, the inductive hypothesis is valid for $j = m+1$. It is therefore valid for $j = 0,1,2,3,\ldots$.

Step 4: From the inductive hypothesis it follows that $x_{37} = a + 37d$. ♦

Example 3:

Given: A list $\boldsymbol{c} = (c_1,\ldots,c_m)$ of *m* distinct positive integers and a positive integer *C.*

Task: Determine whether there is a list $\boldsymbol{x} = (x_1,\ldots,x_m)$ of *m* non-negative integers such that $x_1c_1 + \cdots + x_mc_m = C$.

Step 1: All that needs to be done to generate a family of related problems out of the

given problem is to view the right hand side of the equation, *C*, as a parameter, call it *p*, taking values in the set $P = \{0,1,2,...,C\}$. This family of parametric problems can be stated as follows: given the list *c* and some *p* in *P*, determine whether there is there a list of non-negative integers $\boldsymbol{x} = (x_1,...,x_m)$ such that $x_1c_1 + \cdots + x_mc_m = p$.

Step 2: Let *v*(*p*):="*yes*" be the (correct) answer to question for the given value of *p*. Then it is easy to verify that *v*(0) ="*yes*" and that *v*(*p*) ="*no*" for all 0 < *p* < *c*(*k*), where *c*(*k*) = *arg min* $\{c_j : j = 1,2,...,m\}$. It is also not too difficult to conclude that that if *p* is an element of *P*, then *v*(*p*)="*yes*" if and only if there is some *j* in $M := \{1,2,...,m\}$ such that *v*(*p* – *c*(*j*)) ="*yes*". Hence,

$$v(p) = \begin{cases} \text{"}yes\text{"} & , \quad if \;\; p = 0 \\ \text{"}no\text{"} & , \quad if \;\; 0<p<c(k) \\ \text{"}yes\text{"} & , \quad \text{if } \; v(p - c(j)) = \text{"}yes\text{"} \;\; \text{for some j} \in M \\ \text{"no"} & , \quad \text{otherwise} \end{cases} \tag{5}$$

This is a typical *DP functional equation* for decision problems.

Step 3: Set *v*(0) ="*yes*" and then use (5) to compute the values of *v*(*p*) for $p = 1,2,...,C$ – in this order.

Step 4: The value of interest is *V*(*C*), which we obtain in the last iteration of Step 3. ♦

Example 4: Shortest path problem (see *Graph and Network Optimization*).

Given: A square matrix *t* of non-negative numbers representing the direct travel times between *m* cities and a pair of cities (*s*,*d*). Note that *t*(*i*,*j*) denotes the travel time from city *i* to city *j* along the direct link between these to cities. It is assumed that there is exactly one direct link between any pair of cities.

Task: Compute the length of the shortest (time-wise) path from city *s* to city *d* as well as the shortest path itself.

Note: it is convenient to let *t*(*j*,*j*) =∞ for all *j* in *M*, where *M* is the set of cities.

Step 1: Let $j = 1,2,...,m$ be the index representing the cities under consideration and with no loss of generality assume that the objective is to go from city *s=1* to city *d=m*. It is also assumed that the duration of a tour is additive: that is the travel time from city *A* to city *C* via city *B* is equal to $t(A,B) + t(B,C)$.

One simple way to generate a family of related problems out of this problem is to view the destination city, *d*, as a parameter taking values in the set $M := \{1,2,...,m\}$. The parametric problem is then: what is the length of the shortest path from city *1* to city *j*, for any *j in M ?* There are *m* such problems to solve.

Step 2: Let *f*(*j*) denote the length of the shortest path from city *1* to city *j*. Then clearly, by definition *f*(*1*) = 0. Furthermore, it is not too difficult to verify that for any city *j* other than *1* we have

$$f(j) = t(i, j) + f(i) \tag{6}$$

for some city *i* in *M* such that $i \neq j$.

This is so because by definition *f*(*j*) is the length of some path from city *1* to city *j*. Hence, for $i \neq j$ the value of *f*(*j*) must be equal to the sum of two travel times associated with some city $i \neq j$: the travel time from city *1* to city *i* and the travel time from the city *i* to city *j*. But clearly in order to make the sum as small as possible, it is necessary to go from city *1* to city *i* along the shortest path.

For the same reason, it is also clear that city *i* will satisfy (6) if and only if it will yield the smallest possible value (over all cities other than *j*) for the right-hand of (6). Hence,

$$f(j) = \min_{i \neq j} \{t(i, j) + f(i)\}, 1 \leq j \leq m. \tag{7}$$

This is a typical *DP functional equation* for optimization problems.

Step 3: Solving this equation can be tricky because there is no apparent order in which the values of *f*(*j*) can be computed. Suffice it to say, however, that in some cases the solution procedure is simple. For example, in some cases it is easy to label the cities in such a way that an optimal tour is always "increasing" in the sense that if *j* is a successor of node *i* on the optimal path then *I* < *j*. In such cases (7) can be simplified to

$$f(j) = \min_{i < j} \{t(i, j) + f(i)\}, i \leq j \leq m. \tag{8}$$

To solve the functional equation, set $f(1) = 0$ and then use (8) to determine the value of *f*(*j*) for $j = 2, 3, ..., m$—in this order.

Step 4: The problem of interest is associated with *j* = *m* so to determine the length of the shortest path we have to determine the value of *f*(*m*). Furthermore, we also have to determine the shortest tour itself. The Recovery of optimal solutions will be examined in the discussion on DP algorithms.

As illustrated by these examples, DP is indeed a "general purpose" approach to problem solving. Henceforth we shall focus on its usage as an *optimization method*, namely a method for solving optimization problems. Furthermore, it will be convenient to consider first optimization problems of the following form:

$$z^* := \underset{\mathbf{x} \in \Omega}{opt}\, q(\mathbf{x}) \quad , \quad \Omega \subseteq \mathbf{D}^k \tag{9}$$

where Ω is a real valued function on some non-empty finite set ***D.***

-
-
-

Bibliography

Bellman R. (1957). *Dynamic Programming*, Princeton, NJ: Princeton University Press. [Bellman's first book on dynamic programming.]

Bellman R. (1984). *Eye of the Hurricane: An Autobiography*, Singapore: World Scientific. [Bellman's autobiography with many anecdotes on the history and development of dynamic programming.]

Bertsekas D.P. and Tsitsiklis J.N. (1996). *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific. [This book describes an approximation method based on neural networks and dynamic programming with applications to complex problems of planning, optimal decision making, and intelligent control.]

Brown T.A. and Strauch R.E. (1965). Dynamic programming in multiplicative lattices, *Journal of Mathematical Analysis and Applications* **12**, 364–370. [An outline of a dynamic programming approach to the solution of preference order sequential decision processes.]

Denardo E.V. (1968). Contraction mappings in the theory of dynamic programming, *SIAM Review* **9**, 165–177. [A formal exposition of the use of contraction mappings in the solution of functional equation of dynamic programming associated with non-truncated sequential decision processes.]

Ibaraki T. (1987). *Enumerative Approaches to Combinatorial Optimization*, Basel, Switzerland: J.C. Baltzer AG. [An extensive analysis of dynamic programming as a combinatorial optimization method, including a comparison with branch and bound.]

Mitten L.G. (1964). Composition principles for synthesis of optimal multistage processes, *Operations Research* **12**, 414–424. [An approach to dynamic programming based on monotonicity properties of the objective function.]

Ross S.M. (1983). *Introduction to Stochastic Dynamic Programming*, New York: Academic Press. [An introduction to the use of dynamic programming in the context of stochastic sequential decision processes.]

Sniedovich M. (1992). *Dynamic Programming*, New York: Marcel Dekker. [Modern interpretation of Bellman's conception of dynamic programming, including the principle of optimality.]

Verdu S. and Poor H.V. (1987). Abstract dynamic programming models under commutativity conditions, *SIAM Journal of Control and Optimization* **25**(4), 990–1006. [Analysis of a dynamic programming approach to problems that are not optimization problems.]

Biographical Sketch

Moshe Sniedovich is an operations research scholar with research and teaching interests in dynamic programming, global optimization, interactive computing and modeling and the OR/WWW interface. He was born in Israel in 1945, where after completing his military service, he obtained a degree in Agricultural Engineering (Technion, 1968). He then worked for four years as a National Planning Engineer in the Ministry of Agriculture before commencing his Ph.D. studies at the University of Arizona. Upon completing his Ph.D. (1976), he spent one year at Princeton University and two years at the IBM Thomas J. Watson Research Center as a Post Doc. He then worked for ten years at the CSIR in Pretoria, South Africa. In 1989 he joined the Department of Mathematics at the University of Melbourne, Australia.

His publication record consists of one book and more than 80 articles. Currently he is a Vice President (representing the Asia Pacific region) of the International Federation of Operational Research Societies (IFORS).

A copy of his full CV can be found at www.ms.unimelb.edu.au/~moshe/.