Introduction to the 2010 Edition

Stuart Dreyfus

In this classic book Richard Bellman introduces the reader to the mathematical theory of his subject, dynamic programming. His goal is to show how multistage decision processes, occurring in various kinds of situations of concern to military, business, and industrial planners and to economists, are amenable to mathematical analysis. Written during the infancy of high-speed, large-capacity digital computers, it is understandable that he takes as his goal the mathematical deduction of the structure of optimal decision policies for such problems.

He brilliantly accomplishes this for a surprising variety of situations involving both deterministic and stochastic processes, continuous as well as discrete-stage evolution, and finite as well as infinite problem duration.

Even while demonstrating his impressive mathematical ingenuity, Bellman informs the reader in his preface that research is already underway on the computational solution of problems for which general solution structures are unattainable. He clearly believes that dynamic-programming problems can be mathematically intractable yet, when approached with formulational ingenuity, yield results of practical value. I was privileged to join him in this effort.

We illustrated and attempted to popularize this computational application of dynamic programming using largely military and industrial planning problems of the kind faced by members of our operations-research community. Despite our best efforts, however, surveys of applied practitioners in our area regularly, and painfully for us, showed dynamic programming to be used much less than our planning competitor, linear programming. In the real world of operational planning, the number of state-variable values needed to describe any particular situation that might be encountered during a sequential planning process had frequently turned out to be too large for computational treatment. Dynamic programming seemed to have fallen victim to what Bellman has called the "curse of dimensionality."

One might well wonder, why reprint this introductory volume if dynamic

programming has been shown to be of limited value in the field of its birth? I discovered a very good reason when I began my research leading to this introduction. Computational dynamic programming, I learned, had found its rightful home away from home in the subfield of bioinformatics called computational genomics and in many areas of computer science. There, the number of state variables is small, usually one or two, and the payoffs are large when measured by usefulness. Since linear programming can claim only a few applications to engineering design beyond its traditional operations-research problem domain, my veil of inferiority has lifted.

Two classes of situations account for a significant number of important dynamic-programming applications. The first is termed "sequence alignment." A prototypical alignment problem is: given sequences A of n data elements and B of m data elements ($m>>n$), find the possibly perturbed subsequence of B that best matches A. Depending on the application, allowable perturbations are d in number and include such modifications of B as inserting duplicates of elements, deleting elements, sometimes even changing elements. For each particular application a cost structure must be created whereby the cost of the mismatch of A and a candidate perturbed subsequence of B is defined, and a cost of the perturbations required to produce the subsequence must be found so that the algorithm does not play too fast and loose with its perturbations. Then the perturbed subsequence of B with minimum total cost is sought. This problem, when solved by dynamic programming, can be viewed as an m-stage problem with its single state variable assuming n values at each stage and with d decisions per state and stage, so computation of the solution is of order mnd. Algorithms of this sort constitute the most valuable optimization-guaranteed mathematical tools of computational genomics, wherein the genome is searched for genes or for other significant sequences.

This area of application, since it often has medical significance, would have greatly pleased Richard Bellman. During his post-RAND career as a professor at the University of Southern California he was, among his three departments, a professor in the medical school, due to his passion for finding medical applications for mathematics.

Sequence alignment has also proved useful in automatic speech recognition. Here, dynamic-programming time-warping algorithms seek, among the sequences of elements representing a dictionary, the one best matching the representation of a spoken word. Perturbations of the spoken word are allowed by duplication or deletion of elements to account for the fact that a user's speech speed may shorten or lengthen all or part of a word compared to the dictionary's version. These are just two examples of the large class of sequence-alignment dynamic-programming applications. Several of the ad-

ditional areas of applications of dynamic programming to be noted below are alignment situations.

A second large class of dynamic-programming algorithms frequently falling outside the usual purview of operations research involves hidden Markov model problems. This is a statistical application of dynamic programming where one member of a set of Markov process models, each with its known state transition probabilities, is assumed best to explain a particular observed sequence of states. What is observed, however, is not the true sequence, but an error-corrupted representation; hence the term "hidden" in the topic name. This type of problem arises in a popular approach to automatic speech recognition where each word in the dictionary is represented, not by a given sequence, but by a Markov model. Other recognition problems involving handwriting, musical scores, and topics in bioinformatics have been treated by applying dynamic programming to hidden Markov models.

A partial list of other areas employing computational dynamic programming includes the determination of optimal play in chess endgames, the optimal order for performing chain matrix multiplication, relational database query optimization, edge-following methods used in Photoshop and in artificial vision schemes, and finding the most pleasing justification and hyphenation of text in programs such as TeX. I was recently intrigued by an announcement that a cipher presented to President Jefferson by a mathematician friend who believed it to be undecipherable had been decoded after two hundred years by a method computationally feasible in Jefferson's day although vastly accelerated by computer. The culminating step in the decryption process, unavailable until recently, turned out to be a dynamic-programming sequence-alignment algorithm. Can linear programming top that?

Quite different from the above ingenious ways of using computational dynamic programming is a current approach to machine learning that is called "temporal difference reinforcement learning," or TDRL. A computer is required to learn an optimal, or at least a very good, decision policy controlling either a deterministic or a stochastic sequential decision process by means of process observations. Reward, which is to be maximized, is obtained either at the end of the process, or during the evolution of the process, or both. The algorithm to be used to solve the problem is not told the rules for the decision-dependent evolution of the state or for the determination of reward. All that the training algorithm allows is the exploration of various decisions in various states and the observation of the results. These results that inform the computer may be produced by actually observing real-world situations or they may be computer-produced using rules that are inaccessible to the algorithm being trained.

The conventional approach of physical scientists or engineers would be to observe a great many realizations of the sequential decision process using various decision policies and to thereby gather information that might include probability data. This information would then be used to create and refine a model of the situation that would be used to determine a good or optimal decision policy, perhaps using conventional dynamic programming. (In fact, on page xxii of the preface to this book, observation-based model building is described as a necessary step in learning to predict and thereby control.)

What is intriguing about temporal difference learning is that model building is shown to be unnecessary for learning to predict and control. What can be learned directly from experience is a successive approximation of the optimal-value function and the optimal-policy function of dynamic programming. The term "temporal difference" in the name of this machine-learning procedure refers, during learning of these functions by observation of each step in a process realization, to the difference between the left-hand side of the mathematical equation expressing Bellman's principle of optimality involving a reward-to-go function at time t and the right-hand side during the realization of a step of the sequential decision process that involves that function at time $t + 1$. This difference, for a deterministic process, is zero when the optimal reward-to-go values on both sides are correct and when the decision is optimal. During learning, if it is not zero, this temporal difference becomes the basis of improving the estimates of both optimal reward-to-go value and decision. (For a stochastic process, the difference between the left-hand side and the *expected* value of the right-hand side is zero and the same procedure is applied at each step of a realization of the process.)

This model-free learning of optimal reward-to-go values and optimal decisions, based on observations of realizations of the sequential process as various decisions are explored, was, as far as I know, never contemplated by Bellman. It is now an active area of research in the machine-learning community.

Remarkably, a school of behavioral neuroscientists studying how real brains learn skilled behavior based on experiences has speculated, supported by a growing body of evidence, that both human and lower-animal brains use this model-free TDRL dynamic-programming approach. I have no doubt that Richard Bellman would have felt both pleased and honored to learn that evolution may, before him, have discovered and exploited his beloved principle of optimality.