Jul23 180

The word **dynamic** was chosen by Richard Bellman in 1950 to describe the process of solving problems where one finds the best decisions one after another in a recurrent process. The reason for choosing this name for the method was two-fold. On the one hand, it captured the time-varying aspect of the problems. On the other hand, it sounded impressive. The word **programming** referred to the use of the method to find an optimal program, in the sense of a military schedule for training or logistics, meaning mathematical optimization.

By 1953, Bellman refined this to the modern meaning, referring specifically to nesting smaller decision problems inside larger decisions, and the field was thereafter recognized by the IEEE as a systems analysis and engineering topic.

"I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was

to use the word "programming". I wanted to get across the idea that this was

dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."

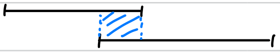
Dynamic Programming

Weighted Interval Scheduling

n requests $1, \dots, n$

$i \to (s_i, f_i, \underset{\text{weight}}{v_i})$ $\qquad v(S_c) = \sum_{j \in S_c} v_j$

$i < j \Rightarrow f_i < f_j$



$P(j) = \max\{ i \text{ s.t. } i < j \ \& \ i \cap j = \emptyset \}$

O_j is opt solution for $1, \dots, j$ requests

Opt is $v(O_j)$

Pr 1 $\quad \text{Opt } j = \max\{ v_j + \text{Opt } p(j), \text{Opt}(j-1) \}$

Proof

1) $j \in O_j \qquad O_j \setminus \{j\} \subseteq \{1, 2, \dots, p(j)\}$

$\qquad\qquad\qquad\qquad V_j + \text{Opt } p(j)$

2) $j \notin O_j \qquad O_j = O_{j-1}$

$\text{Opt } j = \text{Opt}(j-1)$

```
Compute Opt j :
    If j = 0, then Opt j = 0
    else Opt j = max { compute Opt p(j) + V_j , compute Opt(j-1)}
    End If
```

Pr 2 $\quad$ Algorithm Compute Opt n computes Opt n

Proof $\quad$ By induction

$n = 1$

Assume Opt$(n-1)$ show Opt n

Pr 3 $\quad T_A(n) = O(2^n)$

Proof $n = 2 \quad T_A(2) = 2$

$T_A(n-1) = 2^{n-1}$

$T_A(j) = 2^{j} \quad \forall j < n$

$T_A(j) = 2^{p(j)} + 2^{j-1}$

$= O(2^{j-1}) + O(2^{j-1})$

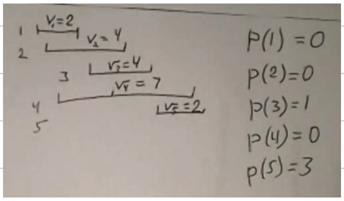
$= O(2^{j})$

Memorizing $M(j) = \text{Opt } j$

$M[0, 1, 2, 3, \dots, n]$

M- Compute Opt j

```
If $j = 0$, then $\text{Opt } j = 0 = M[0]$
Else
If $j \neq n$ then $\text{Opt } j = \max\{ \underset{M[p(j)]}{\underset{\|}{\text{M- comp Opt } p(j)}} + v_j , \underset{M[j-1]}{\underset{\|}{\text{M- comp Opt } (j-1)}} \} = M[j]$
End If
End If
```



$O_1 = \{1\} \quad \text{Opt } 1 = 2$

$O_2 = \{2\} \quad \text{Opt } 2 = \max\{0 + v_2 , \overset{2}{\text{Opt } 1}\} = 4$

$O_3 = \{1, 3\} \quad \text{Opt } 3 = \max\{2 + 4 , \overset{4}{\text{Opt } 2}\} = 6$

$O_4 = \{4\} \quad \text{Opt } 4 = \max\{0 + 7 , \overset{6}{\text{Opt } 3}\} = 7$

$O_5 = \{1, 3, 5\} \quad \text{Opt } 5 = \max\{\text{Opt } 3 + v_5 , \text{Opt } 4\} = 8$

$6 + 2 > 7$

Subset sum Problem

n items $\{1, 2, \dots, n\} = X$

weights $\{w_1, w_2, \dots, w_n\}$

$Q \subseteq X$ s.t. $\sum_{i \in Q} w_i \leq W$ & $\sum_{i \in Q} w_i$ max

1) By decreasing weight

$W = 2n$

$\{\frac{W}{2} + 1, \frac{W}{2}, \frac{W}{2}\}$ $\quad \sum = \frac{W}{2} + 1$

$\sum = 2n = W$

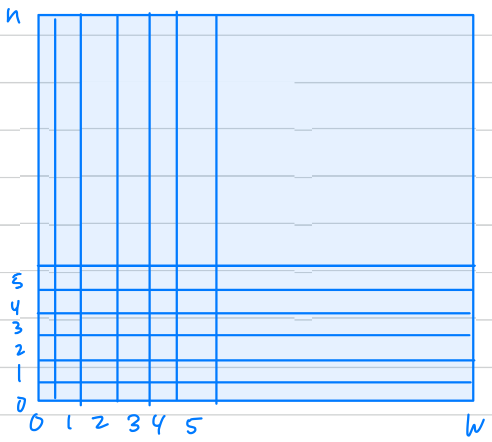
2) By increasing weights

$\{1, \frac{W}{2}, \frac{W}{2}\}$ $\quad \sum = \frac{W}{2} + 1$

$\sum = W = 2n$

Pr 1 If $w_i > w$, then $Opt(j, w) = Opt(j-1, w)$
$\forall j > i$

Array $M[0, 1, 2, \dots, n; 0, 1, 2, \dots, W]$



Pr 2 $Opt(i, w) = \max\{ Opt(i-1, w), w_i + Opt(i-1, w-w_i)\}$

Proof $i = 1$ w_1

1) $w_i \in Q_i$ $Opt(i-1) + w_i \leq W$

$-w_i$ $-w_i$

2) $w_i \notin Q_i$ $Opt(i-1, w)$

Subset-Sum (n, w)

```
Put $M[0, v] = 0$ for $\forall$ $v = 0, 1, 2, \dots, w$

    For $i = 1, 2, 3, \dots, n$
        If $w < w_i$, then $Opt(i, w) = Opt(i-1, w)$
        Else $Opt(i, w) = \max\{ Opt(i-1, w), w_i + Opt(i-1, w-w_i)\}$
       End If
    End For
```