Dynamic Programming: Efficient & Reliable Optimization

The name dynamic programming refers to a specific style of algorithm. It provides a way to take a dataset and optimize a value within a constraint. Dynamic programming isn't the only solution. We could brute force the problem by comparing every possible arrangement of data and choosing the one with the best outcome. The simple solution would quickly get out of hand though, it's complexity O(2^n).

So?

Dynamic programming shortcuts the problem by dividing it into a number of smaller problems. It solves the smaller problems, finds the best solution, and then applies only the best solutions to the larger problem. This way we can cut out a huge number of useless operations without losing accuracy.

Example

Let's take a look at this in action. Let's say you're visiting your local bottle shop. They have some beers you haven't tried before. How can you get an optimal beer experience with a limited budget? Say $10?

Beer	Price	Rating
Three Philosophers	$7	3.8
Sawtooth Ale	$1	3.2

Tap 4 Meine Festweisse	$3	3.6
Westmalle Tripel	$4	3.85
Tired Hands Shambolic	$10	3.88

We're going to calculate the highest total beer rating we can get for each dollar without repeating beers. We use that total to calculate later totals.

	$1	$2	$3	$4	$5	$6	$7	$8	$9	$10
Three Philosophers	0	0	0	0	0	0	3.8 T	3.8 T	3.8 T	3.8 T
Sawtooth Ale	3.2 S	3.2 S	3.2 S	3.2 S	3.2 S	3.2 S	3.8 T	7.0 S,T	7.0 S,T	7.0 S,T
Festweisse	3.2 S	3.2 S	3.6 F	6.8 S, F	6.8 S, F	6.8 S, F	6.8 S, F	7.0 S,T	7.0 S,T	7.4 T,F
Westmalle	3.2 S	3.2 S	3.6 F	3.85 W	7.05 S,W	7.05 S,W	7.45 F,W	10.68 S,F,W	10.68 S,F,W	10.68 S,F, W
Shambolic	3.2 S	3.2 S	3.6 F	3.85 W	7.05 S,W	7.05 S,W	7.45 F,W	10.68 S,F,W	10.68 S,F,W	10.68 S,F, W

We travel through each row, starting at the top. Each box holds the highest total rating we can afford, either from the current row, or from the row above. By solving for lower numbers of beers at lower dollar amounts first we limited the options we needed to think about for each later box.

To brute force a solution we'd need to perform 32 separate comparisons, so it seems like a manual comparison might be better, but what if you added another beer? The brute force solution would need 64 operations to our 60. The algorithm we built with dynamic programs scales at O(10n), so for any significant number of data points it is far more efficient.

What's the catch?

If our choice of item changes the cost or value of other items the system breaks. We're building up a big answer out of smaller answers, and we can't rely on those smaller answers if they might shift out from under us. It would be like building on quicksand.

Dynamic programming isn’t a general use algorithm either. Each problem needs to be dealt with individually. Your target and limits have a big impact on how you construct the algorithm.

Some other uses:

Dna analysis
Text string analysis
Beat tracking
The Selinger algorithm for database query optimization