Dynamic programming - divide and conquer optimisation

During the recent HackerRank's competition *101 Hack 53* I stumbled upon a very nice »problem« which involves solving a DP recurrence, at which point I got stuck and couldn't do it fast enough. It turns out that the recurrence is solvable using the *Divide and conquer optimisation technique*, which I now describe in detail. (»TLDR«)

For illustrative purposes, I will elaborate throughout on the competition **problem** (optimal one-dimensional clustering):

> *Consider a given set of n points $P = \{x_i \mid i = 1, \ldots, n\}$. We would like to find a partition of P into at most k clusters/intervals $C_1, C_2, \ldots$, such that the sum of the within-cluster distances $E := \sum_{i=1}^{n} |x_i - \mu_i|^2$ is minimised, where μ_i is the mean of the cluster to which x_i belongs.*

The divide and conquer optimisation is applicable to dynamic programming problems of the form:

$$dp[i, j] = \min_{k<j}(dp[i-1, k] + cost(k+1, j)) \qquad (1)$$

where the cost function satisfies a **certain property** which we will describe later (Or equally well for the case of maximisation). Let's straight away verify that our problem is indeed of this form. First of all, consider the points $x_1, x_2, \ldots, x_n$ to be in sorted order. We observe that there must an optimal clustering which will consist of consecutive intervals of these points (otherwise, we could swap two points between two clusters and improve the error term E). Let $dp[i, j]$ denote the minimal possible E when partitioning $x_1, \ldots, x_j$ into exactly i clusters. We can then recurse on the size of the (last) cluster, to which the point x_i belongs to. This gives
$dp[i, j] = min_{k<j}(dp[i-1, k] + \sum_{l=k+1}^{j} |x_l - \mu_j|^2)$.

Before we dive into optimising the computation of dp, let's quickly give a fast way of computing the cost term $\sum_{l=k+1}^{j} |x_l - \mu_j|^2$. We can rewrite:

$$cost(k+1,l) := \sum_{l=k+1}^{j} |x_l - \mu_j|^2 = \sum_{l=k+1}^{j} (x_l^2 - 2x_l\mu_j + \mu_j^2) = \sum_{l=k}^{\cdot}$$

$$= \sum_{l=k}^{\cdot}$$

$$= \sum_{l=k}^{\cdot}$$

$$= \sum_{l=k}^{\cdot}$$

After precomputing the consecutive sums of x_is and their squares (which takes linear time), we can obtain any cost(k + 1, l) in constant time.

```
S_1, S_2 = [0], [0]
for x in xs:
    S_1.append(S_1[-1] + x)
    S_2.append(S_2[-1] + x * x)

def cost(k, j):
  d = (S_1[j] - S_1[k]) * 1.0
  return S_2[j] - S_2[k] - d * d / (j - k)
```

It is not hard to solve the DP problem (1) in $O(IJ^2)$ time (where I is the range of i and J range of j); it suffices to simply loop over i, j, k. This gives us a first, naïve $O(n^2k)$ solution to our clustering problem.

```
dp = [[float('inf') for _ in range(N+1)] for _ in range(
dp[0][0] = 0
for i in range(1, K+1):
    for j in range(1, N+1):
        for k in range(j):
            dp[i][j] = min(dp[i][j], dp[i-1][k] + cost(k
print(dp[K][N])
```

Let's see what the **property** for divide and conquer applicability is. Define $\alpha(i,j)$ be the smallest index k in the DP recursion (1) which minimises $dp[i,j]$. We can use the divide and conquer optimisation if, for a fixed i, the sequence $\alpha(i,0), \alpha(i,1), \alpha(i,2), \ldots$ is *monotonic* (that is, non-increasing or non-decreasing). The idea behind optimisation is that this will narrow the range $k < j$ which we need to check for computing $dp[i,j]$.

We now prove that that our clustering problem satisfies this property. In particular, that

$$\alpha(i,0) \leq \alpha(i,1) \leq \cdots \leq \alpha(i,n)$$

Proof. As we will see, a *sufficient condition* for the above monotonicity is that the cost function satisfies the so-called **Quadrangle inequality**: For all $a \leq b \leq c \leq d$ the following holds:

$$cost(a, c) + cost(b, d) \leq cost(a, d) + cost(b, c)$$

In our case, this is equivalent to (Assuming that $a < b < c < d$, in the case of some inequalities being equalities, the statement of quadrangle inequality holds more obviously):

$$\sum_{l=a}^{c} x_l^2 - \frac{(\sum_{l=a}^{c} x_l)^2}{c-a} + \sum_{l=b}^{d} x_l^2 - \frac{(\sum_{l=b}^{d} x_l)^2}{d-b} \leq \sum_{l=a}^{d} x_l^2 - \frac{(\sum_{l=a}^{d}$$

$$\frac{(\sum_{l=a}^{c} x_l)^2}{c-a} + \frac{(\sum_{l=b}^{d} x_l)^2}{d-b} \geq \frac{(\sum_{l=a}^{d} x_l)^2}{d-a} + \frac{(}{}$$

$$\frac{(s_1+s_2)^2}{l_1+l_2} + \frac{(s_2+s_3)^2}{l_2+l_3} \geq \frac{(s_1+s_2+s_3)^2}{l_1+l_2+l_3}$$

where we made the appropriate substitutions s_1, s_2, s_3 for the sums, as well as $l_1 = b-a,\ l_2 = c-b,\ l_3 = d-c$. Moreover, the fact that the sequence $\{x_i\}$ is ordered constraints us with $\frac{s_1}{l_1} \leq \frac{s_2}{l_2} \leq \frac{s_3}{l_3}$ (these are just means of the three parts). Now, a black magic insight allows us to make two further assumptions. First, notice that in our case, the inequality $cost(a, c) + cost(b, d) \leq cost(a, d) + cost(b, c)$ was invariant under shift of $\{x_i\}$ (we may add the same fixed number to each term x_i without changing the inequality) and we thus may assume that the sum $\sum_{l=a}^{d} x_l = s_1 + s_2 + s_3 = 0$. This immediately entails $s_1 \leq 0$ and $s_3 \geq 0$. If we had $s_1 = 0$, the inequality (2) would hold trivially and hence we consider only $s_1 < 0$.
Moreover, notice that the inequality (2) is *homogeneous* (it holds for s_1, s_2, s_3 if and only if it holds for multiples ms_1, ms_2, ms_3). Thus we may further assume that $s_1 = -1$. So putting our assumptions together ($s_1 + s_2 + s_3 = 0; s_1 = -1; s_2 = 1 - s_3$), the inequality (2) is equivalent to:

$$\frac{s_3^2}{l_1+l_2} + \frac{1}{l_2+l_3} \geq \frac{(1-s_3)^2}{l_2}$$

$$s_3^2(l_2^2 + l_2l_3) + l_1l_2 + l_2^2 \geq (1 - 2s_3 + s_3^2)(l_1l_2 + l_1l_3 + l_2^2 + l_2l_3)$$

$$0 \geq l_3(l_1+l_2) - 2s_3(l_1+l_2)(l_2+l_3) + s_3^2$$

$$2s_3 \geq \frac{l_3}{l_2+l_3} + \frac{l_1}{l_1+l_2}s_3^2$$

$$2s_3 \geq \frac{l_3}{l_2+l_3} + \frac{l_1}{l_1+l_2}s_3^2$$

The given inequalities on the right immediately give the inequality on the left. Following the equivalences upwards, the quadrangle inequality is proven. □

Now only one question remains - why is the Quadrangle inequality on the cost function sufficient to assert $\alpha(i,0) \leq \alpha(i,1) \leq \cdots \leq \alpha(i,n)$? *Proof*. We need to show that for all j we have $\alpha(i,j) \leq \alpha(i,j+1)$. Suppose it was otherwise. Then there would be two indices b, a which minimise $cost(\_, j+1)$ and $cost(\_, j)$ respectively, such that $b < a \leq j < j+1$. But this directly gives us two relations $cost(b, j+1) \leq cost(a, j+1)$ and $cost(a,j) < cost(b,j)$, which we can sum to obtain

$$cost(b, j+1) + cost(a,j) < cost(a, j+1) + cost(b,j)$$

This directly contradicts the Quadrangle inequality which states $cost(b, j+1) + cost(a,j) \geq cost(a, j+1) + cost(b,j)$. $\square$

Anyhow, how do we use the fact that $\alpha(i,0) \leq \alpha(i,1) \leq \cdots \leq \alpha(i,n)$ to divide and conquer? In the same way as in the binary search! That is, we will check the value in the middle and then conquer on left and right. Suppose that we first compute the value $dp[i,j]$ and it is minimised at index $\alpha(i,j)$. Then for anything on the left of j, $j' < j$, to compute $dp[i,j']$ we only need to check indices k smaller or equal to $\alpha(i,j)$. Similarly, for anything on the right of j, we only need to check indices k greater or equal to $\alpha(i,j)$. Notice that we can indeed *compute the values $dp[i,\_]$ in arbitrary order*, as they only depend on the previous row $dp[i-1,\_]$. This reduces the runtime of our program from $O(IJ^2)$ to $O(IJ \log J)$.

Revisiting our clustering problem, we can write a faster, $O(kn \log n)$ using this.

```
dp = [[float('inf') for _ in range(N+1)] for _ in range(
dp[0][0] = 0
for i in range(1, K+1):
    stack = [(1, N, 0, N)]
    while stack:
        l, r, alpha_l, alpha_r = stack.pop()
        if l > r: continue
        mid = (r + l)//2
        alpha = alpha_l
        # compute the dp value in the middle
        for k in range(alpha_l, min(mid, alpha_r + 1)):
            here = dp[i - 1][k] + cost(k, mid)
            if here < dp[i][mid]:
                dp[i][mid], alpha = here, k
        # conquer
        stack.append((l, mid-1, alpha_l, alpha))
        stack.append((mid+1, r, alpha, alpha_r))
print(dp[K][N])
```

Full source code that passes the problem at HackerRank (at the very least using PyPy 2) is below:
optimal_bus_stops.py

TLDR

If your DP relation of the form
$dp[i,j] = \min_{k<j}(dp[i-1,k] + cost(k+1,j))$ satisfies the Quadrangle inequality

$$\text{For all } a \leq b \leq c \leq d. \quad cost(a,c) + cost(b,d) \leq cost(a,d) + co.$$

you can compute the values of the i-th row in a divide and conquer order (start in the middle, conquer on the left and right) to obtain a $O(IJ \log J)$ solution.

goatleaps@gmail.com Github Instagram HackerRank