CSE530: Algorithms & Complexity
Notes on Lecture 3: Dynamic Programming

Antoine Vigneron

March 6, 2018

1 Why DTW is not a metric

Our example is in one dimension. Take the three sequences of one or two real numbers $A = (0, 0)$, $B = (0)$ and $C = (1)$. Then we have $\mathrm{DTW}(A, B) = 0$ but $A \neq B$, which already shows that DTW is not a metric.

In addition DTW does not satisfy the triangle inequality, because $\mathrm{DTW}(A, C) = 2$, $\mathrm{DTW}(B, C) = 1$ and thus

$$\mathrm{DTW}(A, C) > \mathrm{DTW}(A, B) + \mathrm{DTW}(B, C).$$

In the example above we used '0' three times. But we do not need to use the same point multiple times in order to build such an example. For instance, after a small perturbation, we have $A' = (0, \frac{1}{100})$, $B' = (\frac{2}{100})$ and $C' = (1)$, and then clearly it still holds that

$$\mathrm{DTW}(A', C') > \mathrm{DTW}(A', B') + \mathrm{DTW}(B', C').$$

2 Answer to last question

In this section, we answer the question asked in the last slide of Lecture 3.

2.1 Computing the histogram

One problem with the function Histogram presented in the slides is that it does not compute an optimal histogram, it only computes its cost $\sum_{p=1}^{m} f^*(s_p, e_p)$. Here we show how to compute the histogram itself, that is, we show how to find s_p, e_p, and x_p^* for all the buckets b_p.

Usually, if we can compute an optimal value $g(x^*)$ of a function g by dynamic programming, it is possible to recover an optimal solution x^* with only a small overhead in terms of time and space. There are several ways of doing it. We present one approach below which consists in computing an auxiliary array that allows to reconstruct the solution in linear time after having computed its cost. It is also possible without this auxiliary data structure, but it makes the code longer. Finally, an optimal histogram can be computed through a completely different approach, called *memoization*. I will not cover it in CSE530; it is presented in Chapter 15 of the textbook. The idea is to solve the problem recursively, and store the result of each recursive call in a global array, so that we can reuse these values instead of recomputing them multiple times.

Here is the modified version of Histogram which computes an optimal histogram (Algorithm 1). It uses an auxiliary array $J[i, k]$ which gives the splitting index j, that is, the index j of the last

Algorithm 1 Computing an optimal histogram

function ConstructHistogram($n, x_1, \ldots, x_n, m$)
  if $m \geqslant n$ **then**
    return 0
  $H \leftarrow$ new $n \times n \times m$ array
  $J \leftarrow$ new $n \times m$ array ▷ Auxiliary array J
  $E \leftarrow$ new array of size m ▷ Record s_p for an optimal solution
  $X \leftarrow$ new array of size m ▷ Record s_p and x_p for an optimal solution
  for $i \leftarrow 1, n$ **do**
    for $j \leftarrow i, n$ **do**
      $H[i, j, 1] \leftarrow f^*(i, j)$
  for $k = 2, \ldots, m$ **do** ▷ Computing the optimal cost
    for $i \leftarrow k, n$ **do**
      $H[1, i, k] \leftarrow \min_{j \in \{k-1, \ldots, i-1\}} \Big(H[1, j, k-1] + H[j+1, i, 1] \Big)$
      $J[i, k] \leftarrow \arg\min_{j \in \{k-1, \ldots, i-1\}} \Big(H[1, j, k-1] + H[j+1, i, 1] \Big)$ ▷ Filling J
  $E[m] \leftarrow n$
  for $p \leftarrow m-1, 1$ **do** ▷ Computing the endpoints e_p
    $E[p] \leftarrow J[E[p+1], p+1]$
  $X[1] \leftarrow H[1, E[1], 1]$
  for $p \leftarrow 2, m$ **do** ▷ Computing the values $x_p^* = f^*(e_{p-1}+1, e_p)$
    $X[p] \leftarrow H[E[p-1]+1, E[p], 1]$
  return E and X

element e_{k-1} of the second last bucket in an optimal $k-1$ buckets histogram for $x_1, \ldots, x_k$. Its elements are computed in Line 14 at the same time as we compute the elements of H. Then after the main loop, using these splitting indices, we can reconstruct an optimal solution backwards in $O(m)$ time.

3 Running time

The bottleneck in ConstructHistogram is the first nested loop, which computes $f^*(s,e)$ for all $1 \leqslant s \leqslant e \leqslant n$. Each value $f^*(s,e)$ is computed in time $\Theta(j-i+1)$, so overall it takes $\Theta(n^3)$ time. We will show below how to compute all these values in $O(n^2)$ time. Then the overall running time becomes $\Theta(n^2 m)$, which is a substantial improvement: In practice, the number m of buckets should be much smaller than the number n of input points.

In order to compute $f^*(s,e)$, we first need to compute the optimal value

$$x^*(s,e) = \frac{1}{e-s+1} \sum_{i=s}^{e} x_i.$$

First observe that

$$x^*(s,e) = \frac{1}{e-s+1} \left(\sum_{i=1}^{e} x_i - \sum_{i=1}^{s-1} x_i \right) = \frac{1}{e-s+1} \left(S_1(e) - S_1(s-1) \right)$$

where $S_1(k) = \sum_{i=1}^{k} x_i$. We can compute all the values $S_1(k)$ for $1 \leqslant k \leqslant n$ in $O(n)$ time, using the recurrence relation $S_1(k+1) = S_1(k) + x_{k+1}$. After that, each value $x^*(s,e)$ can be computed in constant time. So we just proved the following.

Lemma 1. *We can compute all the values of $x^*(s,e)$ for $1 \leqslant s \leqslant e \leqslant n$ in time $O(n^2)$.*

We now show how to compute the values $f^*(s,e)$. Let us rewrite the expression of $f(s,e,\hat{x})$.

$$\begin{aligned} f(s,e,\hat{x}) &= \sum_{i=s}^{e} (\hat{x} - x_i)^2 \\ &= \sum_{i=s}^{e} \hat{x}^2 - 2 \sum_{i=s}^{e} \hat{x} x_i + \sum_{i=s}^{e} x_i^2 \\ &= (e-s+1)\hat{x}^2 - 2\hat{x} \sum_{i=s}^{e} x_i + \sum_{i=s}^{e} x_i^2 \\ &= (e-s+1)\hat{x}^2 - 2\hat{x}(S_1(e) - S_1(s-1)) + S_2(e) - S_2(s-1) \end{aligned}$$

where $S_2(k) = \sum_{i=1}^{k} x_i^2$. Using the same approach as for S_1, we can compute all the values of S_2 in $O(n^2)$ time. After that, we can compute each value of $f(s,e,x^*(s,e))$ in constant time using the formula above. We thus obtain the following result.

Lemma 2. *We can compute all the values of $f^*(s,e)$ for $1 \leqslant s \leqslant e \leqslant n$ in time $O(n^2)$.*

So the running time of the first nested loop in Histogram can be reduced to $O(n^2)$. As the second nested loop runs in $\Theta(n^2 m)$ time, we obtain the following result.

Proposition 1. *An optimal m-buckets histogram for a set of n input points can be computed in $O(n^2 m)$ time.*

3.1 Space usage

The function ConstructHistogram uses an $n \times n \times m$ array, which is cubic. However, from the pseudocode, it is clear that we only use a quadratic number of elements: We use $H[i, j, 1]$ for all $1 \leqslant i \leqslant j \leqslant n$, and we use $H[1, i, k]$ for all i and k. So we can improve on this by replacing H with two arrays $H_1[i, j]$ and $H_2[i, k]$. In fact, we do not even need to compute the array H_1 explicitly, as we can compute any value $H_1[i, j]$ in constant time after precomputing $S_1(i)$ and $S_2(i)$ for $i = 1, \ldots, n$. (See previous section.) These two arrays take only $O(n)$ space, and the array H_2 takes $O(mn)$ space, so we obtain the following result.

Theorem 1. *An optimal m-buckets histogram for a set of n input points can be computed in $O(n^2 m)$ time using $O(mn)$ space.*

In practical terms, it could be very important: If the size of the array $H[i, j, k]$ exceeds the RAM, then the algorithm will make disk access and increase the running time by several order of magnitude. It is usually the case that, in dynamic programming, the space usage should be much less than the running time. For instance, here, the space usage becomes quadratic, while the running time is cubic.

Another observation is that we can compute the cost of an optimal histogram using $O(n)$ space only. The idea is that we compute the array $H[1, j, k]$ line by line, so we only need to remember the values $H[1, j, k-1]$ for all j in order to compute the values $H[1, j, k]$ for all j. Thus we only need to record two lines of $H[1, j, k]$, which takes $O(n)$ space. However, after doing this, it is not clear how we can compute an optimal histogram, because the information needed to construct it backwards has been erased.