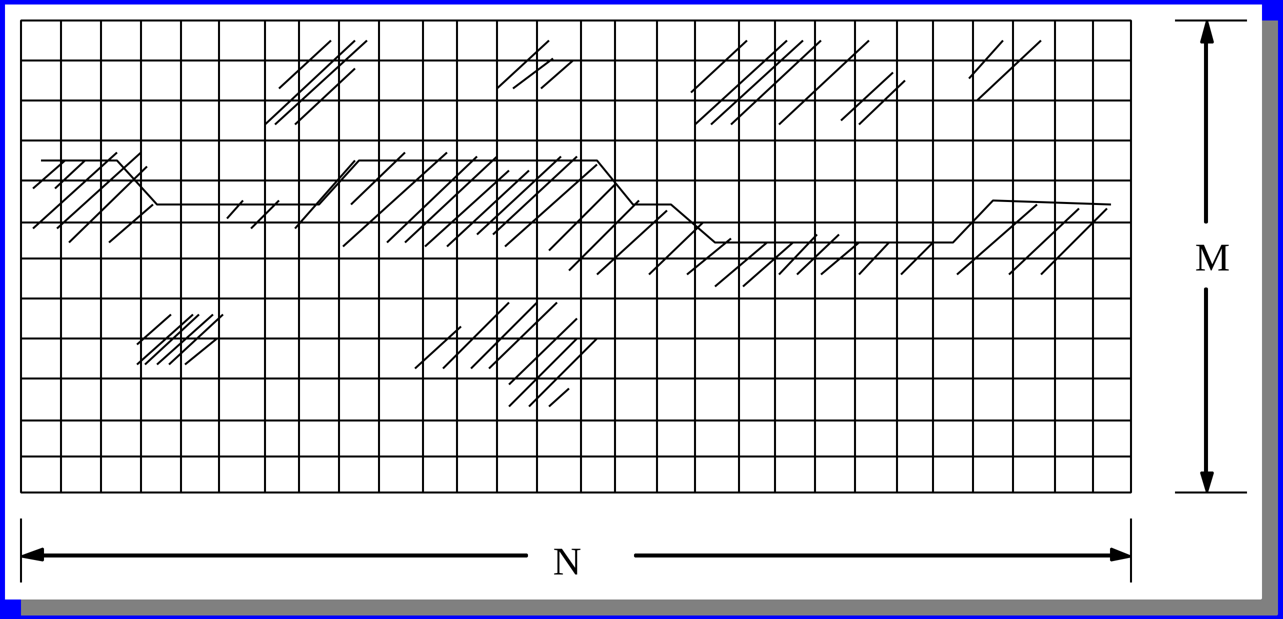
DYNAMIC PROGRAMMING

Boundary detection

Consider an image region of size N x M (see below). Suppose we search a boundary located almost horizontally in that region. Let us represent any possible boundary as a polyline with *N* vertices

$$(p_1,\ldots,p_N) \in P$$



Cost function

Suppose that the boundary we search for is associated with high image intensity, high image gradient and a high degree of continuity (i.e. the curve is smooth). One way of finding the optimal boundary would then be to set up a cost function to be minimized.

Cost function

$$\min \; C_{sum}(P) = \sum_{i=1}^{N} C(p_i)$$

where

$$C(p_1) = -w_1 C_{grad}(p_1) - w_2 C_{int}(p_1)$$

$$C(p_i) = -w_1 C_{grad}(p_i) - w_2 C_{int}(p_i) - w_3 C_{cont}(p_{i-1}, p_i) \quad (i>1)$$

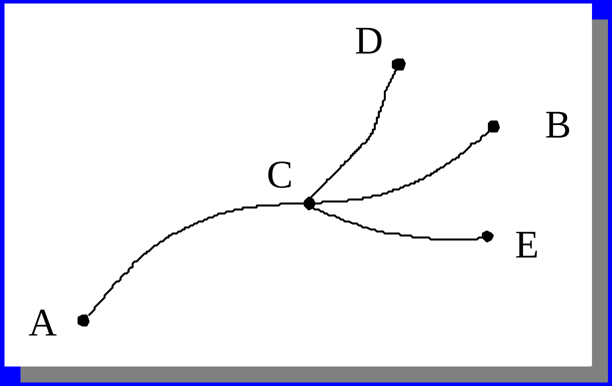
The minimal cost could be found by exhaustive search, i.e. by evaluating the total cost associated with all possible polylines. The complexity of such procedure is $O(N^M)$:

It is clear that we need a more efficient procedure.

Optimization by dynamic programming

Dynamic programming (Bellman and Dreyfus 1962) is based on what is referred to as Bellman's principal of optimality which states that *"the optimum path between two given points is also optimum between any two points lying on the path"*.

If path AB is optimum, then so is AC, no matter how C will go further.



Optimization by dynamic programming

Using dynamic programming it is possible to solve optimisation problems when not all variables in the evaluation function (here, cost function) are interrelated simultaneously. Consider the problem:

$$\max_{x_i} \; h(x_1, x_2, x_3, x_4)$$

If nothing is known about *h*, the only technique that guarantees a global maximum is exhaustive enumeration of all combinations of discrete values *of* (x_1, x_2, x_3, x_4).

Optimization by dynamic programming

Suppose that:

$$h(\cdot) = h_1(x_1, x_2) + h_2(x_2, x_3) + h_3(x_3, x_4)$$

x_1 only depends on *x_2* in *h_1*. Maximize over *x_1* in *h_1* and tabulate the best value of *$h_1(x_{1,} x_2)$* for each *x_2*:

$$f_1(x_2) = \max_{x_1} h_1(x_1, x_2)$$

Since the values of *h_2* and *h_3* do not depend on *x_1*, they need not to be considered at this point.

Optimization by dynamic programming

Continue in this manner and eliminate x_2 and x_3 by computing $f_2(x_3)$ and $f_3(x_4)$ as:

$$f_2(x_3) = \max_{x_2}\left[f_1(x_2) + h_2(x_2, x_3)\right]$$

$$f_3(x_4) = \max_{x_3}\left[f_2(x_3) + h_3(x_3, x_4)\right]$$

So that finally:

$$\max_{x_i} h(\cdot) = \max_{x_4} f_3(x_4)$$

Optimization by dynamic programming

Generalizing the example to *N* variables, where $f_0(x_1)=0$,

$$f_{n-1}(x_n) = \max_{x_{n-1}} \left[f_{n-2}(x_{n-1}) + h_{n\text{-}1}(x_{n\text{-}1}, x_n) \right]$$

$$\max_{x_i} h(\cdot) = \max_{x_N} f_{N-1}(x_N)$$

If each x_i took on 20 discrete values, then to compute $f_N(x_{N+1})$ one must evaluate the maximum for 20 different combinations of x_N and x_{N+1}, so that the resultant computational effort involves $(N\text{-}1)20^2+20$ such evaluations. This is a striking improvement over exhaustive evaluation, which would involve 20^N evaluations of h.

Minimizing the cost function by dynamic programming

Let us now apply this procedure to our problem of minimizing the cost function. If we already know the solution for each of the pixels in column *n*-1, then we can do: for each pixel in column *N*, try to connect it with every pixel in column *n*-1 one by one, choose the best path (and cost) for this pixel. In this way we can get the best solution for each pixel in column *n*. If we already know such solution for each of the pixels in column *n*-2, in the similar way as above, we can get the best solution for each pixel in column *n*-1.

Minimizing the cost function by dynamic programming

We need to find best solutions to arrive each of the pixels in column 2 first. To do that, we rewrite the cost function in the form of a multistage process:

$$
\begin{aligned}
C_{sum}(P) = \; & -w_1 C_{grad}(p_1) - w_2 C_{int}(p_1) \\
& -w_1 C_{grad}(p_2) - w_2 C_{int}(p_2) - w_3 C_{cont}(p_1, p_2) \\
& \ldots \\
& -w_1 C_{grad}(p_{k-1}) - w_2 C_{int}(p_{k-1}) - w_3 C_{cont}(p_{k-2}, p_{k-1}) \\
& -w_1 C_{grad}(p_k) - w_2 C_{int}(p_k) - w_3 C_{cont}(p_{k-1}, p_k) \\
& \ldots \\
& -w_1 C_{grad}(p_N) - w_2 C_{int}(p_N) - w_3 C_{cont}(p_{N-1}, p_N)
\end{aligned}
$$

Dynamic programming algorithm

1 Create a cost accumulation matrix (CAM). Copy the first column of the image into the CAM

2 Starting from the second column in the image, for each point in the column find the optimal solution and store the accumulated cost in the CAM

3 Repeat step 2 until you reach the rightmost column

4 From the node in the rightmost column of the CAM associated with the lowest cost, trace back column-by-column passing through the nodes associated with the lowest accumulated cost. The trace defines the optimal boundary

Simple case

$$C_{sum}(p_i) = C_1(p_i) + C_2(p_{i-1}, p_i)$$

$(p_1 \ldots, p_5 \in P)$

where :

C_1 is presented in the matrix below, and

$$C_2(p_{i-1}, p_i) = 2\Delta y$$

$$C_1: \left|\begin{array}{ccccc} 2 & 2 & 4 & 3 & 1 \\ 4 & 1 & 1 & 2 & 1 \\ 1 & 2 & 5 & 6 & 0 \\ 2 & 2 & 2 & 4 & 2 \\ \hline \end{array}\right.$$

Simple case