trapezoid - solution

This is a dynamic programming task, with the fruitful use of well-known data structures. There are various possible approaches with time and memory complexities O (n^2), O ($n \log^2 n$) and O ($n \log n$).

Every trapezoid is represented by two intervals (one on each horizontal line). We can normalize the coordinates, and get the permutation from 1 to 2n. This can be easily done by sorting the arrays a and b (and c and d) in O ($n \log n$) time, with two additional arrays.

We could make things easier by introducing two dummy trapezoids, one on the left and one on the right side with large coordinates.

Next, we will calculate the size of the maximum independent set. We can define partial ordering by sorting the trapezoids according to the upper left corner a (i). Let *max_ind* (i) denote the size of the maximum independent set of trapezoids $T(1)$, $T(2)$, …, $T(i\text{-}1)$ that contain trapezoid $T(i)$. It simply follows:

max_ind (i) = *max* {1 + *max_ind* (k)},
where $1 \le k \le i$ and T (k) lies completely on the left of T (i)

This produces O (n^2) dynamic programming algorithm.

In order to get O ($n \log n$) solution, we will use a binary indexed tree (cumulative table) data structure. Cumulative tables in logarithmic time perform three operations on the array x:

- for given index k and number m, add m to the value x (k)
- for given index k, calculate the partial sum x (1) + x (2) + … + x (k)
- for given index k, calculate the maximum value among x (1), x (2), …, x (k).

Let *cum_max* be the cumulative table for maintaining the partial maximums. Traverse the coordinates from 1 to 2n, and for each left upper coordinate a (i) calculate *max_ind* (i) based on the maximum value among *cum_max* (1), *cum_max* (2), …, *cum_max* (c (i)), while for the right upper coordinate b (i) write *max_ind* (i) in the cumulative table on the index d (i). The final solution is *max_ind* (n + 1).

The second part is more difficult. The quadratic dynamic programming solution is given in the following pseudo code:

```
num_max_ind (0) = 1;
for i = 1 to n do begin
        num_max_ind (i) = 0;
        for j = 0 to i – 1 do begin
                if ((max_ind (j) + 1 == max_ind (i)) and (a (j) > b (i)) and (c (j) > d (i)) then
                        num_max_ind (i) = num_max_ind (i) + num_max_ind (j);
        end;
end;
```

We will use again cumulative tables for the designing O (n log n) solution. The main problem here is how to manipulate the partial sums. Namely, for each trapezoid $T(i)$, we need to sum the values num_max_ind (j) of those trapezoids $T(j)$ which are entirely on the left of $T(i)$ with additional condition max_ind (j) + 1 = max_ind (i).

This can be done by considering pairs (k, k + 1) of two neighboring values $1 \leq k \leq max\_ind$ (n + 1). Using two additional arrays for coordinates and trapezoid indices, we can traverse trapezoids T (i) from left to right, such that max_ind (i) = k or max_ind (i) = k + 1. First, we need to carefully preprocess all trapezoids and store the coordinates for each pair (k, k + 1) in one array (or array of dynamic arrays) in order to keep memory limit O (n). Therefore, we fill the cumulative table with the values for trapezoids with max_ind (i) = k,and calculate the value num_max_ind (i) for trapezoids with max_ind (i) = k+1. Instead of resetting the cumulative table for each k, we can remove the values by another traversing the array of indices.

Since every trapezoid will be added and removed from the cumulative table, the total time complexity is O (n log n). This reusing of the cumulative table makes this task very interesting.