3. **Line Intersections (Redux)**

Now, suppose you are given a set L of n line segments in the plane, but the endpoints of the segment lie on the unit circle (e.g. defined by the equation $x^2 + y^2 = 1$), and all $2n$ endpoints are distinct. Give an algorithm to compute the largest subset of L in which no pair of segments intersects. To obtain full marks your algorithm should run in time $O(n^3)$[1].

Solution

We solve this problem by way of dynamic programming. There are several ways to do it — one is to give a memoized recursive dynamic programming algorithm which fixes a line and recurses on the "areas" of the circle which remain, such as in the minimum-weight triangulation algorithm. This gives an $O(n^3)$ running time, and the implementation is very similar to that of the minimum-weight triangulation algorithm. A modification of this algorithm runs in time $O(n^2)$, which we will present here.

We use $\ell_1, \ell_2, \ldots, \ell_n$ to denote the n line segments in L. As stated, these line segments all have distinct endpoints and all of the endpoints of the lines lie on the unit circle. Let P be the set of all endpoints of the lines in L. Let p_1 be one of the endpoints of ℓ_1 and re-label p_1 with the integer 1. Then, label the rest of the points with integers in increasing order by starting at p_1 and moving clockwise around the circle. After this relabeling we will have $P = \{1, 2, \ldots, 2n\}$, and each line ℓ in the input can be identified as a pair of endpoints in P. For an example of such an ordering see Figure 2.

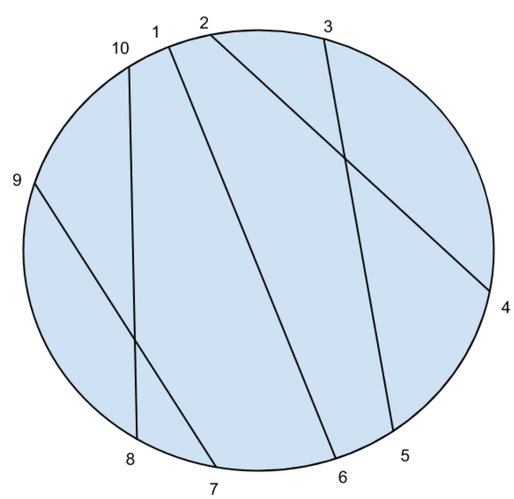

Figure 2: A circle ordering of the line endpoints

Using this new definition of the endpoints we can easily determine if two lines intersect. (From here on, whenever we consider a line ℓ from the input, we will implicitly assume it is defined by a pair of integers from P in increasing order. So the line in Figure 2 with endpoints 1 and 6 will always be written as $\ell = (1, 6)$, not $(6, 1)$.) For suppose that $\ell_1 = (i, j)$, $\ell_2 = (k, l)$, and assume without loss of generality that $i < k$. Then ℓ_1 and ℓ_2 intersect if and only if $i < k < j < l$ — to convince yourself of this, choose two lines in Figure 2 and test it out.

Before we give the recursive definition, first suppose that i, j are two integers with $1 \le i < j \le 2n$. We define the subset of lines $L_{ij} \subseteq L$ as follows: a line $\ell = (a, b)$ is in L_{ij} if $i \le a < b \le j$. For example, in Figure 2, the set L_{16} contains the lines $(1, 6)$, $(2, 4)$, and $(3, 5)$. More generally the set L_{ij} consists of all lines from the input with with both endpoints "between" i and j. Note that $L_{1,2n} = L$.

[1] $O(n^3)$ is not the best running time possible for this problem. I will offer a bonus (up to 5%) for anyone who can beat the $O(n^3)$ algorithm.

We are ready to give the recursive definition used by the algorithm. We make an $n \times n$ array D and for each $1 \leq i \leq j \leq 2n$ we set

$$D[i,j] := \text{the maximum number of non-intersecting line segments in the set } L_{ij}.$$

By our comment above, the maximum number of non-intersecting line segments in the circle will be stored in $D[1, 2n]$.

For the base cases of the recursive definition, for each $i \leq 2n - 1$ we set

$$D[i, i+1] = \begin{cases} 1 & \text{if } (i, i+1) \text{ is a line in the input} \\ 0 & \text{otherwise} \end{cases}$$

and also $D[i, i] = 0$.

As for the recursive definition, we set

$$D[i,j] := \max \begin{cases} D[i+1, j-1] + 1 & \text{if } (i,j) \text{ is a line in the input} \\ D[i+1, k-1] + D[k+1, j] + 1 & \text{if } (i,k) \text{ is a line in the input and } i < k < j \\ D[i, k-1] + D[k+1, j-1] + 1 & \text{if } (k,j) \text{ is a line in the input and } i < k < j \\ D[i+1, j-1] & \text{otherwise} \end{cases}$$

We will prove this recursive definition is correct by induction over $j - i$. If $j - i = 0$ then $j = i$ and $L_{i,i} = \emptyset$, since all of the lines in the input have distinct endpoints. If $j - i = 1$, then $j = i + 1$, and either $L_{i,i+1} = \emptyset$ if $(i, i+1)$ is not a line in the input, or $L_{i,i+1} = \{(i, i+1)\}$ if $(i, i+1)$ is a line in the input. Both of these cases are covered correctly by the base cases of D above. Assume that $D[i, j]$ is correctly defined for all $j - i < m$, and we will prove that it is correctly defined when $j - i = m$.

Let O_{ij} be the largest set of non-intersecting lines in L_{ij}. We want to prove that $D[i, j] = |O_{ij}|$. There are several cases:

Case 1: (i, j) is a line in O_{ij}

In this case, then (i, j) cannot intersect with any other line in L_{ij} since every other line $\ell = (k_1, k_2)$ in L_{ij} satisfies $i < k_1$ and $k_2 < j$ by the distinctness of the endpoints. Thus, $O_{ij} = O_{i+1,j-1} \cup \{(i, j)\}$, and so $|O_{ij}| = D[i + 1, j - 1] + 1$ by the inductive hypothesis.

Case 2: (i, k) is a line in O_{ij} with $k \neq j$.

In this case we must have $i < k < j$. The rest of the lines in O_{ij} must therefore have endpoints between i and k and between k and j. This implies that

$$O_{ij} = O_{i+1,k-1} \cup O_{k+1,j} \cup \{(i,k)\},$$

and so $O_{ij} = D[i + 1, k - 1] + D[k + 1, j] + 1$ by the inductive hypothesis.

Case 3: (k, j) is a line in O_{ij} with $k \neq i$.

This is identical to Case 2, except now we have

$$O_{ij} = O_{i+1,k-1} \cup O_{k+1,j} \cup \{(i,k)\}.$$

By the inductive hypothesis this implies that $|O_{ij}| = D[i + 1, k - 1] + D[k + 1, j] + 1$.

Case 4: There are no lines in O_{ij} with endpoint i or j.

In this this, all of the lines $\ell = (k_1, k_2)$ in O_{ij} must satisfy $i < k_1$ and $k_2 < j$. This means that $O_{ij} = O_{i+1,j-1}$, and so the inductive hypothesis implies that $O_{ij} = D[i+1, j-1]$.

We can therefore express $|O_{ij}|$ as the maximum of all of these four cases. This gives us exactly the expression for $D[i, j]$ given above, and so $D[i, j] = |O_{ij}|$. The proof is complete.

Finally, we give an algorithm (Algorithm 3) implementing our recurrence. Note that finding such an ordering of the endpoints P as we defined it above can be determined in linear time by fixing one of the endpoints arbitrarily, and then doing a clockwise sweep of the circle in polar coordinates. We will therefore assume that the endpoints will be given in this manner.

Algorithm 3 runs in $O(n^2)$ time, owing to the doubly-nested for-loop.

Algorithm 3: Circle Intersection

Input: A list of n lines in the unit circle L, each line with distinct endpoints, and a circular ordering P of the $2n$ endpoints.

Output: The largest set of non-intersecting lines in L.

```
/* D is the semantic array, S is an array helping us building the
   optimal solution, and M is a helper array storing line endpoints    */
let D be an n × n array, initialized to all 0s;
let S be a n × n array, initialized to all ∅s;
let M be a 2n × 1 array, initialized to all 0s;
for each line (i, j) ∈ L do
    if j = i + 1 then
        D[i, i + 1] = 1;
        S[i, i + 1] = {(i, i + 1)};
    else
        M[i] = j;
        M[j] = i;
    end
end
for gap = 2, 3, ..., 2n − 1 do
    for i = 1, 2, ..., 2n − gap do
        j := i + gap;
        t_1 = t_2 = t_3 = 0;
        S_1 = S_2 = S_3 = ∅;
        if M[i] = j then
            t_1 = D[i + 1, j − 1] + 1;
            S_1 = S[i + 1, j − 1] ∪ {(i, j)};
        else
            t_1 = D[i + 1, j − 1];
            S_1 = S[i + 1, j − 1];
        end
        if M[i] = k and i < k < j then
            t_2 = D[i + 1, k − 1] + D[k + 1, j] + 1;
            S_2 = S[i + 1, k − 1] ∪ D[k + 1, j] ∪ {(i, k)}
        end
        if M[j] = k and i < k < j then
            t_3 = D[i, k − 1] + D[k + 1, j − 1] + 1;
            S_3 = S[i, k − 1] ∪ D[k + 1, j − 1] ∪ {(k, j)}
        end
        D[i, j] = max_{i=1,2,3} {t_i};
        index = arg max_{i=1,2,3} {t_i};
        S[i, j] = S_index;
    end
end
return S[1, 2n]
```