Perimeter Product Problem

In this problem, we are given a list of line segments, which each segment has the coordination of its start and end points as: [((x1, y1), (x2, y2)), …], We asked to calculate the multiplicative product of all perimeters of all the individual enclosed areas.
To solve this problem, I have two ideas: 1) Use geometric and find the exact coordination of all original points and also the intersection points we have no information about these intersection points and we have to use geometry and find them. 2) Use graph theory, which I am interested in, and consider each line segment as the edge of the graph, and all points as nodes of the graph. Then, using graph modeling, find all unique loops and sub-loops, and calculate their perimeters. In the following section, I describe this method as my solution to this problem. I coded my solution in Python and tested it with different scenarios successfully.

Solution

In this solution, I considered all points of segments as nodes of a no-directional graph. Also, I considered the line segments as the edges of graph G. Based on my experience with the Networkx package in Python 3, I used the version of 3.3 of it for my solution. I have considered two algorithms for this solution.
The first algorithm, which you can find in the **make_graph.py** file, tried to make a graph based on available input edges (line segments) and nodes (points of line segments). In complex shapes, line segments will have overlap and intersections, but we do not know the coordination of those possible intersections or intersections, maybe a few intersections on one line segment. Therefore, based on only one intersection in the middle of a line segment, we will have one new node and two new sub-segments. Therefore, in this algorithm, first I looked for any possible intersections (one or more) (nodes) on available segments (edges) using the **edge_intersect()** function. This function finds all new intersections (nodes) and adds them to our graph G. The function **create_graph()** created a graph based on previous nodes and additional intersection nodes, added new sub_edges (sub_segments) as new edges to the graph, and removed the original edges (original segments) from the graph, as we are looking for only smallest loop or cycles inside the graph to be unique area. Based on the number of intersections on original segments, this function should be repeated to find all sub-sub-edges and consider a unique graph G. I used a While loop in the main code, **main.py** file, to make sure I find all possible intersections. As soon as did not get any new intersection, the flag of **last_intersection** in the function of **create_graph()** will be active and the while loop will be terminated.
In the second algorithm, **perimeter_cycles.py** file, after creating a full graph of sub-edges and nodes, first I tried to find all unique sub-cycles (sub-loops) inside the graph. Some loops are complete subsets of other large loops, and we have to remove the large loops and only keep the subset of small loops. In some cases, some small loops may have not a complete subset of the large loop, and maybe one node or edge was not covered in the large loop, mathematically. But, we know the small loop is a subset and part of the large loop. To solve this, I defined a threshold, for example, 0.80, and looked for all small loops that subset of large loops at least in 80% of nodes and edges. It makes to be conservative in different complex scenarios. You can find this code in the function of the **approximate_subset()** file. Then considering this solution, I found all unique sub-cycles in

side the graph using the **find_unique_cycles()** function. I repeated this function in the main code to make sure that find any available unique cycles inside the graph. I sorted all loops based on their size from the largest loop to the smallest loop. Then, compared the first (largest) loop with other loops in the for loop. If there is any subset loop, then remove the largest loop from the unique cycles. I repeated to find all possible cycles several times in the while loop in the main code, **main.py**. After finding all unique sub_cycles, using the function of **calculate_perimeter()**, calculated their perimeters separately. After getting all perimeters of all unique sub_cycles, then multiply them to find the answer to the problem as a product of perimeters. To consider the number of digits after the point in the float number, I used the precision parameter and considered it 6 for rounding the results.
I tested this algorithm in different scenarios and got correct answers quickly.

Examples

Here are several different scenarios for testing the algorithms:

```
edges_window_1: Square with the size of windows in each edge=1 (simple square) (Default)
Output = 4.0000
edges_window_2: Square with the size of windows in each edge=2
Output = 16.0000
edges_window_3: Square with the size of windows in each edge=3
Output = 0.1342
edges_window_d_1: Square with the size of windows in each edge =1, plus a diameter (hourglass
shape)
Output = 5.8284
edges_window_d_2: Square with the size of windows in each edge =2, plus a diameter in one window
Output = 23.3137
edges_window_d_3: Square with the size of windows in each edge =3, plus a diameter in six windows
Output = 0.0053
edges_test: A complex and symmetric shape as a test of the algorithm (TEST)
Output = 5.5713

edges_window_1 = [((1, 1), (1, 2)), ((1, 2), (2, 2)), ((2, 2), (2, 1)), ((2, 1), (1, 1))]
edges_window_2 = [((1, 1), (1, 2)), ((1, 2), (2, 2)), ((2, 2), (2, 1)), ((2, 1), (1, 1)), ((1.5,
1), (1.5, 2)), ((1, 1.5), (2, 1.5))]
edges_window_3 = [((1, 1), (1.6, 1)), ((1, 1.2), (1.6, 1.2)), ((1, 1.4), (1.6, 1.4)), ((1, 1.6),
(1.6, 1.6)), ((1, 1), (1, 1.6)), ((1.2, 1), (1.2, 1.6)), ((1.4, 1), (1.4, 1.6)), ((1.6, 1), (1.6,
1.6))]
edges_window_d_1 = [((1, 1), (1, 2)), ((1, 2), (2, 1)), ((2, 1), (2, 2)), ((2, 2), (1, 1))],
edges_window_d_2 = [((1, 1), (1, 2)), ((1, 2), (2, 2)), ((2, 2), (2, 1)), ((2, 1), (1, 1)),
((1.5, 1), (1.5, 2)), ((1, 1.5), (2, 1.5)), ((1, 1.5), (1.5, 2))]
edges_window_d_3 = [((1, 1), (1.6, 1)), ((1, 1.2), (1.6, 1.2)), ((1, 1.4), (1.6, 1.4)), ((1,
1.6), (1.6, 1.6)), ((1, 1), (1, 1.6)), ((1.2, 1), (1.2, 1.6)), ((1.4, 1), (1.4, 1.6)), ((1.6, 1),
(1.6, 1.6)), ((1, 1.4), (1.2, 1.6)), ((1, 1.2), (1.4, 1.6)), ((1.2, 1.2), (1.6, 1.6)), ((1.4,
1.2), (1.6, 1.4))]

edges_test = [((1, 1), (1, 2)), ((1, 2), (1.8, 2)), ((1.8, 2), (1.8, 1)), ((1.8, 1), (1, 1)),
((1.2, 1), (1.2, 2)), ((1.6, 1), (1.6, 2)), ((1, 1.5), (1.6, 1.8)), ((1, 1.3), (1.8, 1.7)),
((1.2, 1.2), (1.8, 1.5))]
```

Codes:

I have three .py files, **main.py** as the main code, and two helper codes: **make_graph.py** and **perimeter_cycles.py**. To run this algorithm you need to follow in two ways:

```
python3 main.py -e edges_window_1
```

or

```
python3 main.py -e "[((1, 1), (1, 2)), ((1, 2), (2, 2)), ((2, 2), (2, 1)), ((2, 1), (1, 1))]"
```

Test results: edges_test

```
hadi@penguin: ~/perimeter_produc
(p312) hadi@penguin:~/perimeter_product/Perimeter_Product$ python3 main.py -e edges_test
============= Step: 1 =============
============= Step: 2 =============
============= Step: 3 =============

Graph Edges: 27
[((1.2, 1.6), (1.2, 2)), ((1, 2), (1.2, 2)), ((1.8, 1), (1.8, 1.5)), ((1.8, 1.5), (1.8, 1.7)), ((1.6, 1.4), (1.8, 1.5)), ((1.2, 1), (1.2, 1
.2)), ((1.2, 1.6), (1.6, 1.8)), ((1.6, 1), (1.8, 1)), ((1.2, 1.4), (1.2, 1.6)), ((1.6, 2), (1.8, 2)), ((1.2, 1), (1.6, 1)), ((1.6, 1.8), (1
.6, 2)), ((1, 1.5), (1.2, 1.6)), ((1, 1.5), (1, 2)), ((1.6, 1.4), (1.6, 1.6)), ((1, 1.3), (1.2, 1.4)), ((1, 1), (1.2, 1)), ((1, 1.3), (1, 1
.5)), ((1.2, 1.4), (1.6, 1.6)), ((1.2, 2), (1.6, 2)), ((1.2, 1.2), (1.6, 1.4)), ((1.6, 1), (1.6, 1.4)), ((1.8, 1.7), (1.8, 2)), ((1.6, 1.6)
, (1.6, 1.8)), ((1.2, 1.2), (1.2, 1.4)), ((1.6, 1.6), (1.8, 1.7)), ((1, 1), (1, 1.3))]

Inntersections: 4
[(1.2, 1.4), (1.2, 1.6), (1.6, 1.4), (1.6, 1.6)]

Unique Cycles: 10
[[(1.6, 1.8), (1.6, 2), (1.8, 2), (1.8, 1.7), (1.6, 1.6)], [(1.2, 1.4), (1.2, 1.2), (1.2, 1), (1, 1), (1, 1.3)], [(1.2, 1.6), (1.2, 2), (1,
 2), (1, 1.5)], [(1.2, 1.6), (1.2, 2), (1.6, 2), (1.6, 1.8)], [(1.2, 1.6), (1.6, 1.8), (1.6, 1.6), (1.2, 1.4)], [(1.2, 1.6), (1.2, 1.4), (1
, 1.3), (1, 1.5)], [(1.2, 1.4), (1.2, 1.2), (1.6, 1.4), (1.6, 1.6)], [(1.8, 1), (1.8, 1.5), (1.6, 1.4), (1.6, 1)], [(1.2, 1), (1.2, 1.2), (
1.6, 1.4), (1.6, 1)], [(1.8, 1.5), (1.8, 1.7), (1.6, 1.6), (1.6, 1.4)]]

Perimeters of Cycles:  [1.123607, 1.123607, 1.323607, 1.447214, 1.294428, 0.847214, 1.294428, 1.323607, 1.447214, 0.847214]

Product of Perimeters: 5.5713

(p312) hadi@penguin:~/perimeter_product/Perimeter_Product$ python3 main.py -e "[((1, 1), (1, 2)), ((1, 2), (1.8, 2)), ((1.8, 2), (1.8, 1)),
 ((1.8, 1), (1, 1)), ((1.2, 1), (1.2, 2)), ((1.6, 1), (1.6, 2)), ((1, 1.5), (1.6, 1.8)), ((1, 1.3), (1.8, 1.7)), ((1.2, 1.2), (1.8, 1.5))]"
============= Step: 1 =============
============= Step: 2 =============
============= Step: 3 =============

Graph Edges: 27
[((1.2, 1.6), (1.2, 2)), ((1, 2), (1.2, 2)), ((1.8, 1), (1.8, 1.5)), ((1.8, 1.5), (1.8, 1.7)), ((1.6, 1.4), (1.8, 1.5)), ((1.2, 1), (1.2, 1
.2)), ((1.2, 1.6), (1.6, 1.8)), ((1.6, 1), (1.8, 1)), ((1.2, 1.4), (1.2, 1.6)), ((1.6, 2), (1.8, 2)), ((1.2, 1), (1.6, 1)), ((1.6, 1.8), (1
.6, 2)), ((1, 1.5), (1.2, 1.6)), ((1, 1.5), (1, 2)), ((1.6, 1.4), (1.6, 1.6)), ((1, 1.3), (1.2, 1.4)), ((1, 1), (1.2, 1)), ((1, 1.3), (1, 1
.5)), ((1.2, 1.4), (1.6, 1.6)), ((1.2, 2), (1.6, 2)), ((1.2, 1.2), (1.6, 1.4)), ((1.6, 1), (1.6, 1.4)), ((1.8, 1.7), (1.8, 2)), ((1.6, 1.6)
, (1.6, 1.8)), ((1.2, 1.2), (1.2, 1.4)), ((1.6, 1.6), (1.8, 1.7)), ((1, 1), (1, 1.3))]

Inntersections: 4
[(1.2, 1.4), (1.2, 1.6), (1.6, 1.4), (1.6, 1.6)]

Unique Cycles: 10
[[(1.6, 1.8), (1.6, 2), (1.8, 2), (1.8, 1.7), (1.6, 1.6)], [(1.2, 1.4), (1.2, 1.2), (1.2, 1), (1, 1), (1, 1.3)], [(1.2, 1.6), (1.2, 2), (1,
 2), (1, 1.5)], [(1.2, 1.6), (1.2, 2), (1.6, 2), (1.6, 1.8)], [(1.2, 1.6), (1.6, 1.8), (1.6, 1.6), (1.2, 1.4)], [(1.2, 1.6), (1.2, 1.4), (1
, 1.3), (1, 1.5)], [(1.2, 1.4), (1.2, 1.2), (1.6, 1.4), (1.6, 1.6)], [(1.8, 1), (1.8, 1.5), (1.6, 1.4), (1.6, 1)], [(1.2, 1), (1.2, 1.2), (
1.6, 1.4), (1.6, 1)], [(1.8, 1.5), (1.8, 1.7), (1.6, 1.6), (1.6, 1.4)]]

Perimeters of Cycles:  [1.123607, 1.123607, 1.323607, 1.447214, 1.294428, 0.847214, 1.294428, 1.323607, 1.447214, 0.847214]

Product of Perimeters: 5.5713

(p312) hadi@penguin:~/perimeter_product/Perimeter_Product$ _
```