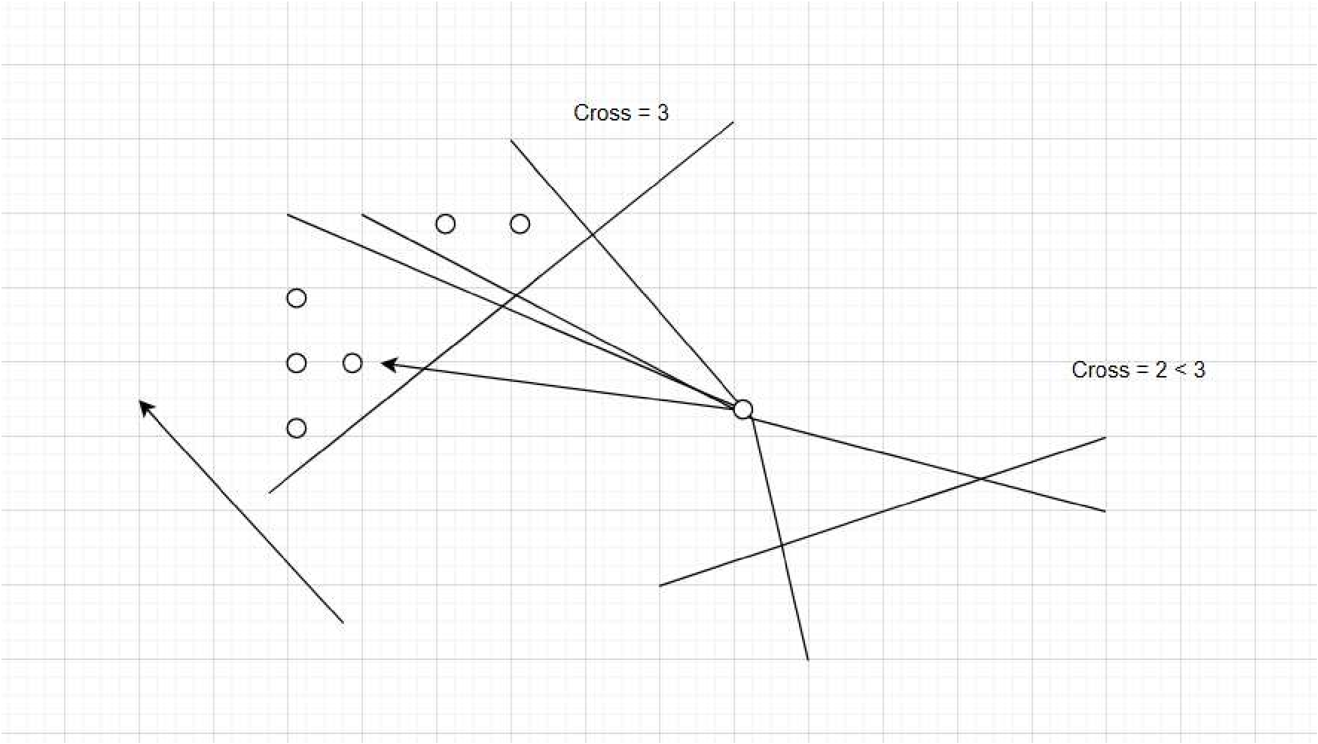
0.

Bottleneck: Checking degree takes O(E^2) or worse.

1.

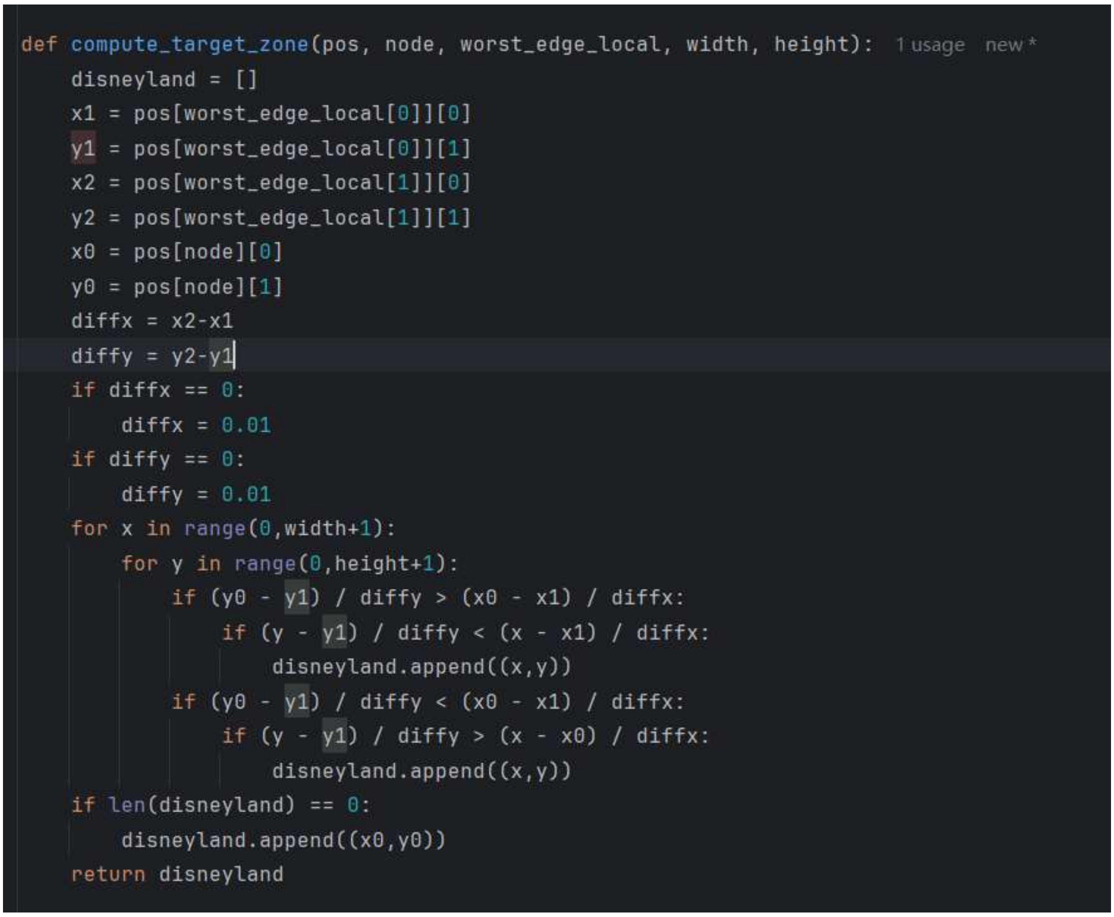
Jeff’s quasi-spring method(new):



Can derive some more complicated metrics, but as what we have, purposed by Jeff last week, we randomly pick a candidate from the other side of the “worse side”.

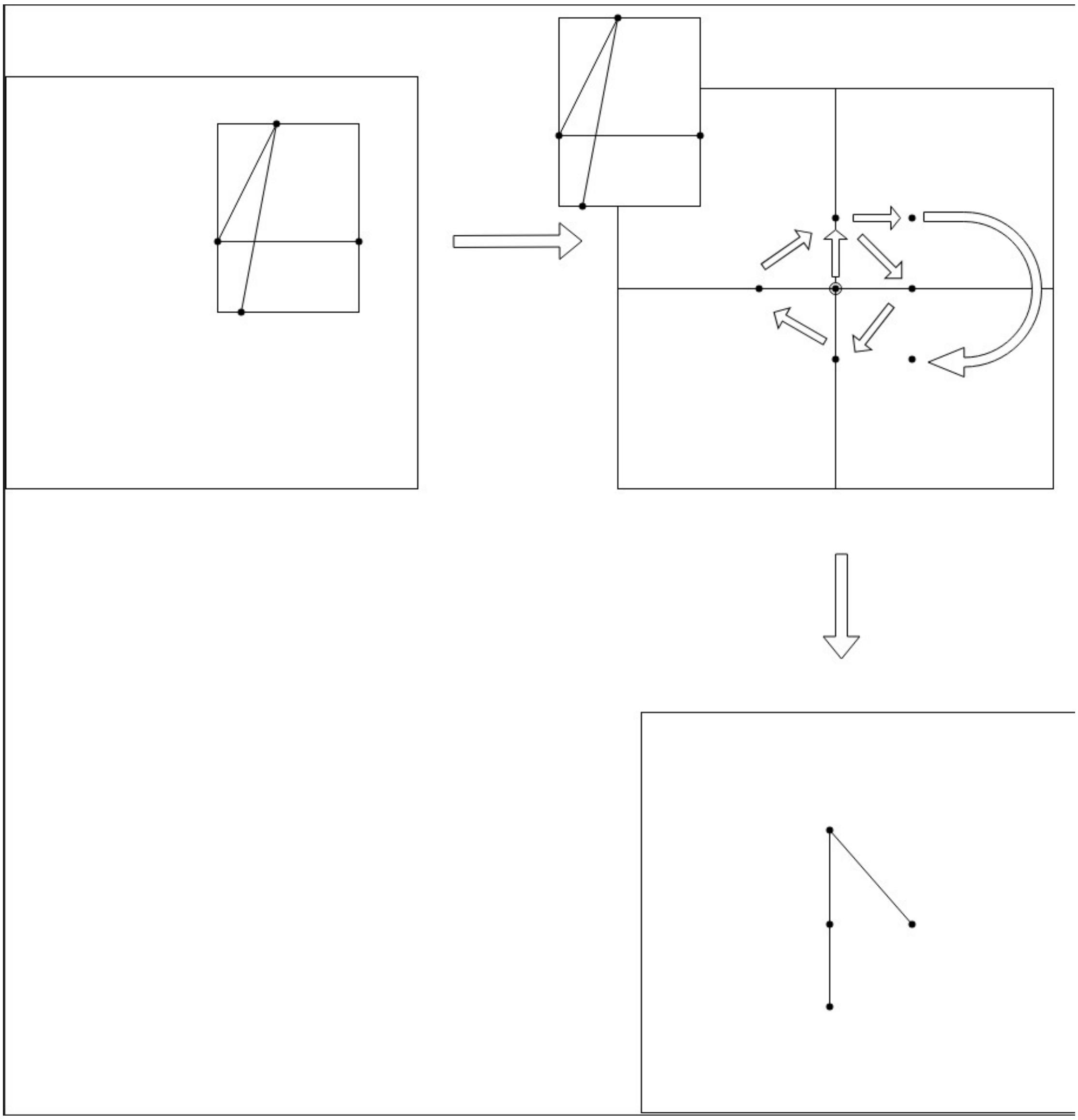
Haven’t run it yet but we can tell this method and it’s simpler variant cost low.

Draft:



```
def compute_target_zone(pos, node, worst_edge_local, width, height):
    disneyland = []
    x1 = pos[worst_edge_local[0]][0]
    y1 = pos[worst_edge_local[0]][1]
    x2 = pos[worst_edge_local[1]][0]
    y2 = pos[worst_edge_local[1]][1]
    x0 = pos[node][0]
    y0 = pos[node][1]
    diffx = x2-x1
    diffy = y2-y1
    if diffx == 0:
        diffx = 0.01
    if diffy == 0:
        diffy = 0.01
    for x in range(0,width+1):
        for y in range(0,height+1):
            if (y0 - y1) / diffy > (x0 - x1) / diffx:
                if (y - y1) / diffy < (x - x1) / diffx:
                    disneyland.append((x,y))
            if (y0 - y1) / diffy < (x0 - x1) / diffx:
                if (y - y1) / diffy > (x - x0) / diffx:
                    disneyland.append((x,y))
    if len(disneyland) == 0:
        disneyland.append((x0,y0))
    return disneyland
```

2. Jeff’s surrounding method(old):



After a few steps, it stucks in a loop.

Can be break but doesn’t seem to be efficient.

3. Other heuristics/mechanisms we have:

3.1 Diameter based(bigger space yields bigger solution space.)

3.2 Bucket sort like limited random climb

3.3 Random exchange/swap

3.4 (Looking for a try): Resolution?