Simulating fungal growth as a discrete network with a decentralized constricted knowledge model

William Tickman

INTRODUCTION

Fungi are all around us, the soil beneath our feet, the air we breathe and, in some capacity,, the bread you eat. Responsible for decomposition and resource distribution, fungi are a key component to the functioning of most environments. Fungi are unique in their ability to form very efficient resource transport pathways through decentralized network decisions. The fungi do not have a central system deciding how to structure it's growth, fungal networks are continuous systems in which each part of the network performs the same function as the rest. Fungi builds these networks while foraging for resources. The structure of growth for these organisms is most easily studied starting the fungi at a single inoculation point and viewing the organism as a two-dimensional system. This is most commonly done in petri-dish style experiments. Fungal Networks start small at a core node and branch outward distributing resources from the core outward to the tips. Initial branches are very thin and tree-like so as to cover more area but as expansion progresses the inner thin branches begin to fuse and form stronger chords, or they are recycled by the system and redistributed. This causes the networks center to be densely connected while the periphery, searching tips, form a much more tree-like branching network. As the network searches further and further for more resources to no avail, it continues to thin out branches and eventually recycle them causing the networks core to become sparse. A particularly difficult aspect of this simulation is how it is spatially distributed and expanding. While not knowing the extent of its own structure it is still able to maintain some sense of direction. There have been many attempts at simulating fungal growth through both continuous and discrete methods. Continuous models are generally much simpler, avoiding hyphae and ranching interactions, and are able to model much larger colonies due to the computational simplicity. These models do not function as the fungi would but instead attempt to mimic its behavior. Discrete models have proven to be very computationally expensive due to the complex structure of the networks formed so are far less often used (Hopkins, 2011). Although very complicated, it is easier to precisely define the functioning of the fungal growth in a discrete model. Because our understanding of these fungal networks is still incomplete, to simulate them, all models must make assumptions. Some simulations are built to ignore the actual growth mechanics of fungi and instead serve the purpose of mimicking overall structure accurately while other models attempt to focus on the actual mechanics of fungi to hopefully, eventually build out larger more accurate networks. I am of the view that modeling the mechanics of fungi is far more useful if not just more interesting. As discrete models have currently been developed there is a heavy reliance on statistical decisions tuned to experimental data. This can cause many issues in trying to model mechanics as it is a sort of hand-wavy random response to unknown functioning. In the mathematical modeling of a fungal network it is important to recognize the flow mechanics of the system. Flow of resources through the system is not just in a particular direction, resources are passed back and forth continually which exerts a pressure on the whole system. This pressure through the network can be modeled as a hydraulic incompressible fluid that requires the system to expand or contract in response (Heaton, Lopez, Maini, Fricker, & Jones, 2010). Viewing the network in this way allows the use of many

fundamental hydraulic and electrical network equations to determine system pressure and flow. Edges of the network can be assigned resistive values based on their attributes.

METHODS

Although spatial environments are continuous, in this simulation the network is stored as a discrete model. Storing the fungal network as a discrete model allows it to be broken up as edges and nodes which makes iterative computations much simpler. For simplicity fungal networks are treated as two dimensional systems in this model. All data is stored into a Networkx graph object. Networkx provides support for high level functionalities such as network measures, data conversions and easy visualization tools. A caveat of using Networkx is its lack of any spatial functionality. As a simulation the model progresses through iterative phases. Each phase is split into two major steps, structural and spatial, between each occurs a synchronous update to the whole network. The first step handles the distribution of resources and growth of the network based on network attributes and structure. Because the first step ignores the spatial component of the network this can result in new edges overlapping old edges without any connection between the two edges. The second step resolves all spatial interactions within the network utilizing the Shapely python package. Shapely provides a robust framework for spatial analysis and operations. Converting the Networkx graph to a Shapely object simplifies finding and resolving edge intersections and concatenating similar edges and nodes. To minimize the assumptions made about fungal network functioning, the data stored about the network has been limited to edge width, edge resistance to growth, edge length, node coordinate location and node food supply amount. The edges comprise the majority of the functioning of the network accounting for all of the distributed resources. Nodes serve the purpose of harvesting food from food sources and immediately distribute those resources to neighboring edges. Beyond food harvesting nodes contain spatial location data and connect edges. The resources used in a single edge is equivalent to the edge width times edge length. The transfer of resources between edges is currently modeled to be 100% efficient meaning that all resources are converted to edge width growth or new hyphal growth without loss, this will be discussed further in later sections. In addition to the fungal network there is the substrate which it grows on. The substrate map is stored as a set of Shapely objects and includes information regarding food sources, in further implementation the substrate can also contain information on physical barriers and resource sinks. Upon initialization of the simulation, nodes are created within the food sources of the substrate, this is congruent to *in vivo* inoculation of food sources. Here begins the iterative phases.

The structural step begins with the harvest of food, iterating through each node in the network to check if the point lies within the substrate maps food source. The distribution of food allocated towards hyphal growth or adding mass to existing edges is dependent on the feedback pressure of the system. This feedback pressure is determined by the resistance of the edges to accepting flow, in this model this is determined by the sum of neighboring edge widths but in a more complete model may include acknowledgements to the edge's plasticity. The plasticity of an edge is the inverse of its resistance to growth. In distributing food towards adding mass to existing edges, the food is distributed proportional to the edge's width with respect to the sum of all neighbors' widths with slight random variation. This causes wider edges to accept more resources while smaller edges are slower to accumulate resources. In the first phase no edges exist so all resources are distributed to hyphal growth. New edges are created as hyphal growth until all harvested food is used. Although there is a lot of parameterization possible for adjusting the width and length of new edges created, the only input variable to be included should be the amount of resources available for new growth. Any other inputs would be non-local knowledge which is forbidden in the functioning of this simulation. To create a new hypha first the direction and length must be determined, I have implemented the direction to be entirely random and for the length to be a random normally distributed value with an upper bound and standard deviation proportional to the food

supply. This limits a hyphae's growth length to smaller values over a single step while still allowing for larger variability if the food supply is in excess. The hyphae's width is based on a random normal distribution in which the mean and maximum possible value are proportional to the maximum width given the food available and predetermined length (Maximum Width = Food Available /Edge Length). There are many nuanced complexities to hyphal growth that can be implemented beyond random normal distributions, but this model provides a solid framework that can be added to in the future. The variability in hyphal growth is supported by the concept of pressures effect on mass flow towards hyphal growth (Heaton, Lopez, Maini, Fricker, & Jones, 2010). After all food has been harvested and distributed to edges, the main body of the structural phase occurs within edge-edge resource transfer. Iterating through each edge in the network local decisions are made to remove resources from the edge and transfer those resources to new hyphal growth or existing edges. Each edge recycles its own resources and distributes them to surrounding edges, if an edge does not receive more resources than it distributes over the entire phase then its width will decrease and eventually the edge may fully recycle itself. The Resource transfer amount for any particular edge is limited by the total resources held within the edge and proportional to its relative width with respect to its' connecting neighbors. The distribution of these resources between hyphal growth and existing edges is modeled as a normal distribution with a mean of 20% and standard deviation of 5% for resources allocated towards hyphal growth. This normal distribution of resources is not empirically based but acts as a working placeholder for more empirically accurate future works. Similar to the distribution of food, resources going to existing edges are distributed proportional to edge width and hyphal growth originates at some random point along the edge. After all resources have been distributed, to reduce runtime significantly, hyphae containing less than 0.001 resources are removed and recycled back into the network. Hyphae of this size are considered resource insignificant however the multitude of edges left at this size makes them troublesome in solving the spatial interactions of the network so they are removed.

The second step of the phase involves correcting all spatial interactions between edges. If two edges overlap or cross then they must also touch and be able to directly transfer resources between each other. The first part of this process is resolving all edge intersections. The Networkx graph must be converted to a Shapely object to perform all spatial functions. Shapely objects are not able to hold any attributes so consistent mapping of edges between the Networkx graph and Shapely network must be maintained. Networkx edges are converted to Shapely LineString objects by taking the coordinates of the nodes at the endpoints of each edge. Iterating through each edge, intersection points are found. From the list of intersection points found, all intersecting edges with those points must be determined and correlated back to the Networkx graph to retrieve attributes. Once two or more intersecting edges have been properly identified, new edges are created, in the Networkx graph, going from the intersection point to the previous edge endpoints, the previous edge is deleted and its attributes are distributed to the new edges. All edge attributes are maintained to be the same value except length which is recalculated. As note this process allows hyphae to grow over and through existing growth. In resolving all intersections many nodes have been added, some of which are so close together that they could be considered the same node. A small buffer is created around each node, nodes with intersecting buffers are concatenated to a new single central(midpoint location) node maintaining all connecting edges of the original nodes, the old nodes are deleted. This step introduces parallel edges into the graph between similar nodes. For a particular pair of nodes all parallel edges are concatenated into a single edge; edge weights are added. There is a clear distinction between this method and the neighbor-sensing model in that this models' buffer is not a continuous feature so the node does not take input from anything outside the immediate buffer, maintaining the individual nodes constricted knowledge view.

Each phase iteration represents a time step in fungal growth. As time progresses the food supply shrinks, this is adjusted by decreasing the harvest rate between phases. Each phase is visualized using Networkx draw capabilities with standardized edge widths and overlayed on the substrate map. For the purpose of this simulation each phase is run manually so that observations can be made about the changes between each phase.

An imperative component of simulating fungal networks is comparing the results to empirical networks. Thankfully a large dataset of fungal networks has been made available (Lee, Fricker, & Porter, 2016). Each network in the dataset is formatted as a MATLAB file of an edge adjacency matrix and a node coordinate matrix. The naming scheme for each file can be obtained from the list_of_fungal_networks.xlsx file which describes the fungi type and methods used in the experiment. Each file was read into python and converted to a Networkx graph to make high level network measures and comparisons easier.

RESULTS

The simulation was able to successfully branch outwards from an initial inoculation site, find food and form a dense cluster around the food, and build stronger chords between food sources. The simulation was unsuccessful in mimicking the pattern of structure often seen in fungal networks. The network was unable to properly recycle edges in a way that would cause the networks core to become more sparsely connected while still branching outwards at the peripheries. I believe that with some fine tuning of the parameters surrounding the allocation of resources between hyphal growth and resource transfer to existing edges this simulation can provide much better results. As this simulation currently stands it is not worthwhile comparing to empirical fungal models. Beyond visual inspection there are no network measures useful to evaluating the failure of the simulation. Although my work did involve evaluating empirical fungal networks that work will not be discussed as it provided no assistance in tuning this simulation. In Figure 1, the red shapes indicate food sources. In this run of the simulation the fungi was initialized to two inoculation points within the outer food sources and between each phase the harvest rate of food was cut in half. In figure 1 A and B the simulated network is observed to have found the middle food source and has begun branching new hyphal growth from there. In images C-E the network is observed to have fully connected the upper two food sources to the lower one to create a single connected network. Image E and F depict the strengthening of the connection between the three food sources. Image G shows only a subset of edges in the network which have the largest edge width. In this view it is clear the edges directed between food sources have grown to be the largest. This is a crucial step in forming efficient pathways and suggests promising potential for this model. From Fig. 1:F it is evident that improvements must be made to the algorithm responsible for concatenating nearby nodes. While the abundance of hyphal growth surrounding food sources is empirically accurate the lack of resource transfer to existing edges is an issue that needs to be solved. This model would benefit the most from an improvement to the resource distribution decision between hyphal growth and existing edges. Although this simulation may not appear visually similar to any fungal network it does share many similar characteristics. The simulation was able to strengthen efficient pathways between food sources while being restricted to an extremely local view for all decisions. All network growth decisions were made from a particular edge or node with only knowledge of the sum of all connected edge widths. With that limitation in mind, it is quite impressive this model was able to properly strengthen efficient pathways.

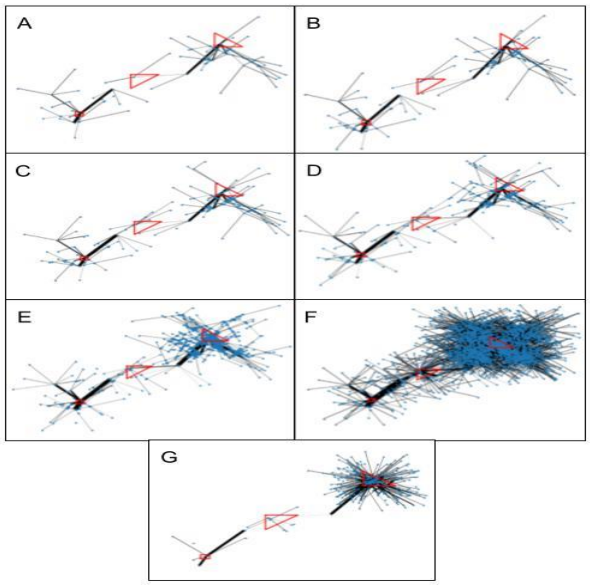

Figure 1: two inoculation sites with three food sources

DISCUSSION

As this model currently stands there is very little mathematical modeling of flow involved. This simulation has been built as a framework that is very easy to build upon. The introduction of hydraulic flow dynamics into the model seems to be an obvious next step. To properly model growth due to pressure it is necessary to create a metric to describe the plasticity of each edge in the network to determine which edges will expand/contract or where hyphal growth may occur. The plasticity is the inverse of an edge's resistance to growth, it may be proportional to an edges width but a key aspect of an edges plasticity is the bidirectional flow of resources. The bi-directional flow of resources is an important metric that ignores the total volume of resource transport and instead measures how resources are passed back and forth. An edge that is only transporting resources in a single direction is unlikely to be well interconnected with the rest of the network so the fungi will refrain from growing that chord at a high rate. For edges with more balanced flow in both directions this is a good indicator that either the edge is located centrally to the network or it is between multiple food sources. In this case it is more probable that the network should expand along this edge, this edge would have a high plasticity. The plasticity of an edge will naturally decrease from lack of use as well. This model currently distributes resources to neighboring edges based proportionally to the edges width however, with the introduction of a plasticity measure the distribution of resources to edges will be much more empirically accurate. This model has made many assumptions of fungal network mechanics, none more egregious than the separation of structural growth from spatial interactions. In separating these two steps, through the structural growth phase, edges are able to grow over each other without regard and often edges will grow backwards into the core of the network. Attempting to avoid assumptions the direction of hyphal growth was set to be random and while this may be proximally accurate, there is some evidence to suggest over larger steps that there is some underlying factor that results in a normal distribution of hyphal growth angle relative to its parent edge (Hopkins, 2011)[2.1.2]. The implementation of normally distributed hyphae branching angles to the existing simulation would be quite trivial but there are many concerns to adopting this statistical method because it would circumvent the known functioning mechanics of the fungi. A limitation of the iterative process for growing hyphae is that often over a single step a hypha may grow over the top of and past an existing edge. In a real fungal network, the hyphae would grow into the existing edge and resources would flow between, it is possible this local pressure increase could cause a new hyphae to originate on the adjacent wall of the chord as if the hyphae had just continued to grow through. Because the structural growth and spatial processes are separated this issue cannot be solved as the hyphae is growing but must be dealt with afterwards. One solution would be to cut the hyphae at the intersection point with the chord and distribute the cut portions resources back into the network sourced from the node location. This method would likely prove to be very difficult in identifying new hypha segments that fit these

criteria while not removing already existing hyphae. This change to the model is likely not necessary as hyphae growing through chords is an empirically supported phenomena that is within the scope of the simulation to properly handle and recycle back into the network. Another spatial assumption that was made was the buffer size around nodes to concatenate nodes. When two nodes are very close together these nodes are fused to create a single node. The proximity of these nodes to warrant fusing together was not empirically based but was necessary for decreasing computational costs. In experimental fungi networks, the edges are attached along a short section not just a point. This section is dynamic along the attached chord and moves as resources flow through it. The resources would eventually flow along the more efficient path within the attachment section pushing the two "nodes" closer together (Heaton, et al., 2012). While there is justification for fusing nearby nodes the process to model resource flow to slowly fuse the nodes together is far too slow and computationally expensive. Fusing nearby nodes allows for the fusing of cords within the simulation without resorting to features assumed in the Neighbor-sensing model which presumes to sense nearby mycelium to fuse nearby branches (MESˇKAUSKAS, FRICKER, & MOORE, 2004). This simulation treats every edge as a storage of resources. Resources do not flow through edges they are stored in edges and then redistributed based on the function defined as ResourceTransferAmount. This allows edges to be iterated through without global concern of overall resource flow. The resource transfer amount is scaled to the resources held within the edge and is proportional the edges relative width compared to neighbors. This cause wider edges to distribute a higher proportion of their stored resources than a thinner connecting edge would. While the assumptions of how many resources an edge can transfer being proportional to it's stored resources is accurate there are far more extensive methods that can be used like local pressure and the plasticity of the edge to determine the recycling rate of an edge. What I consider to be the most significant flaw of this simulation is the allocation of resources between hyphal growth and existing edges. The division of this allocation is set as a random normal distribution heavily favoring growth as edge width accumulation. This allocation would benefit from further hydraulic modeling and the inclusion of parameters such as edge widths and plasticity. While this simulation does make plenty of assumptions nearly all of these assumptions are given as random normal distributions which is similar to the functioning of many other basic fungal simulations (Hopkins, 2011). It is reasonable that with extensive calibration of the distribution of values for point buffer size, food harvest rate, resource allocation and edge growth distribution that this model can better simulate the growth patterns of fungal networks, but it is likely more worthwhile implementing more scientifically based measures first.

Bibliography

Gillies, S. (2020). *Shapely User Manual.* Retrieved from Shapely: https://shapely.readthedocs.io/en/stable/manual.html

Hagbefrg, A., Schult, D., & Swart, P. (2021). Networkx. Retrieved from https://networkx.org

Heaton, L., Lopez, E. L., Maini, P. K., Fricker, M. D., & Jones, N. S. (2010). *Growth-induced mass flows in fungal networks*. Retrieved 5 15, 2021, from http://eprints.maths.ox.ac.uk/974/1/298.pdf

Heaton, L., Obara, B., Grau, V., Jones, N., Nakagaki, T., Boddy, L., & Fricker, M. D. (2012). Analysis of Fungal Networks. *Fungal Biology Reviews*. Retrieved from https://doi.org/10.1016/j.fbr.2012.02.001

Hopkins, S. (2011). *A Hybrid Mathematical Model of Fungal Mycelia: Tropisms,.* Retrieved from https://core.ac.uk/download/pdf/6117416.pdf

Islam, M., Tudryn, G., & R., B. (2017). *Morphology and mechanics of fungal mycelium.* Sci Rep. Retrieved from https://doi.org/10.1038/s41598-017-13295-2

Lee, S. H., Fricker, M. D., & Porter, M. A. (2016). *Mesoscale analyses of fungal networks as an approach for quantifying phenotypic traits.* doi:10.1101/006544

MESˇKAUSKAS1, A., FRICKER, M. D., & MOORE, D. (2004). *Simulating colonial growth of fungi with the Neighbour-Sensingmodel of hyphal growth.* Retrieved from https://doi.org/10.1017/S0953756204001261

Appendix

All code is available at https://github.com/thewilltotick/FungalNetworkSimulations but some has been included below for ease of viewing.

```
#coordinate = tuple, food = num
#returns new node coordinates, weight, length, food used
def hyphalGrowth(coordinates,food):
    if(food < 0.4):
        MaxL = 2
    else:
        MaxL = food#ADJUST VALUE for maximum length relative to food
    x = coordinates[0]
    y = coordinates[1]
    #creates bimodal distribution \/
    xn = x + (np.random.choice([-1,1], size = 1)*ranged_normal(1, 0.7, 0.2, MaxL))[0]
    #ajust values for mean hyphal growth length
    yn = y + (np.random.choice([-1,1], size = 1)*ranged_normal(1, 0.7, 0.2, MaxL))[0]
    coord2 = (xn,yn)
    Elength = edgeLength(coordinates, coord2)
    assert(Elength != 0)
    MaxW = food/Elength
    if(MaxW <= 0.3):#prevents infinite edges added
        weight = MaxW
    else:
        weight = ranged_normal(MaxW/2, 0.3, 0.1, MaxW)[0]#min val=> still okay to add small edges
    foodN = food - (Elength*weight)
    assert(weight > 0)
    return (coord2, weight, Elength, foodN)
```

```
#evenly distributes resources to edges proportional to edge weights
def porpEdgeGrowth(G, edgeNum,  food, edgeWSum):
    if( food < 0.2):
        G.edges[edgeNum]['weight'] = float((food/ G.edges[edgeNum]['length']) + G.edges[edgeNum]['weight'])
        assert(G.edges[edgeNum]['weight'] > 0)
        return 0
    else:
        assert(food >0)
        MaxW = (food/G.edges[edgeNum]['length'])
        assert(MaxW >0)
        p = (G.edges[edgeNum]['weight']/edgeWSum)* MaxW
        Wadd = ranged_normal(p, 0.2, 0, MaxW)[0]
        G.edges[edgeNum]['weight'] = Wadd + G.edges[edgeNum]['weight']
        assert(G.edges[edgeNum]['weight'] > 0)
        return food - Wadd*G.edges[edgeNum]['length']
```

```
#given a graph and a specific edge number returns availabel food resources to be transfered.
#transfer amount is porportional to the current edge weight divided by sum of neighbor edge weights
def ResourceTransferAmount(G, Edge):
    Available = G.edges[Edge]['weight'] * G.edges[Edge]['length']
    SumNeighborWeights = sum([x[2] for x in list(G.edges(Edge[0], data = 'weight'))]) + sum([x[2] for x in list(G.edges(Edge[1], data = 'weight'))]) - G.edges[Edge]['weight']
    transfer =(Available*(G.edges[Edge]['weight']/SumNeighborWeights))
    #transfer
    G.edges[Edge]['weight'] = ((Available - transfer)/G.edges[Edge]['length'])
    return transfer #units are Weight x Length
```

```
#Resolves all spatial intersections of edges in graph GS and returns new graph
#splits edge and distributes edge attributes, new edge is added to graph and old edges are removed
def ResolveIntersections(GS, nodeNum):

    GLines = []
    invertNodesDict = {v: k for k, v in dict(GS.nodes(data = 'coord')).items()}
    invertEdgeDict = {v: k for k, v in dict(GS.nodes(data = 'coord')).items()}
    #convert current graph to shapely edge lines
    GCoordList = list(dict(GS.nodes(data = 'coord')).values())
    for i  in GS.edges():
        GLines.append(LineString([(nx.get_node_attributes(GS, 'coord')[i[0]][0],nx.get_node_attributes(GS, 'coord')[i[0]][1]),
                                  (nx.get_node_attributes(GS, 'coord')[i[1]][0],nx.get_node_attributes(GS, 'coord')[i[1]][1])  ]))

    IntersectLines = MultiLineString(GLines)
    for line in IntersectLines:
        IntersectionsMP = []
        #find intersections of each individual line
        NLines = gpd.GeoSeries(MultiLineString([i for i in GLines if i != line]))
        IntersectionsMP = NLines.unary_union.intersection(line)
        if(IntersectionsMP.is_empty):
            pass

        #coordinate list of intersection points
        if IntersectionsMP.geom_type == 'MultiPoint':
            InterCoords = [(p.x, p.y) for p in IntersectionsMP] #works for multiPoint
        elif IntersectionsMP.geom_type == 'Point':
            InterCoords = list(NLines.unary_union.intersection(line).coords)#works for point
        #resolve every intersection
        for point in InterCoords: #check if nodes already exist in networkx graph
            if(point not in GCoordList): #if node doesn't exist split line, find neighbor nodes, remove/add edges
                GCoordList += point #new point exists, prevents duplicates
                nodeNum = nodeNum +1
                GS.add_node(nodeNum, food = 0, coord=point) #add node to networkx
                #find all intersecting linestrings with new point
                for zine in IntersectLines:
                    if(zine.intersects(Point(point))):#Lines to be split
                        nc1 = zine.coords[0] #endpoint of line/networkx node
                        n1 = invertNodesDict[nc1]#networkx node labels found
                        nc2 = zine.coords[1]
                        n2 = invertNodesDict[nc2]
                        Eattr = GS.edges[n1,n2] #get existing edge attributes
                        GS.add_edge(n1,nodeNum, Eattr)#create new edges
                        GS.edges[n1,nodeNum]['length'] = edgeLength(nc1, point)
                        GS.add_edge(n2,nodeNum, Eattr)
                        GS.edges[n2,nodeNum]['length'] = edgeLength(nc2, point)
                        GS.remove_edge(n1,n2)#remove existing edge
                        #don't really need to split in shapely so don't
    return nodeNum
```

```
# G is graph with node attributes 'coord' and edge attribute 'weight'
#FoodMap is a Shapely multiPolygon containg polygons with food
def plotSim( G, FoodMap):
    for poly in FoodMap:
        plt.plot(*poly.exterior.xy, color = 'red')
    nx.draw(Gsim, pos =Gsim.nodes(data = 'coord'), width=standardWidth(Gsim) , node_size = 2)
    plt.show()
```

```python
#view Empirical networks
def drawMatNetworks( filename):
    print("fungal network:", filename)
    matC = scipy.io.loadmat('fungal_MATLAB\Conductance' +filename)
    positionsC = dict()
    for i in range(len(matC['coordinates'])):
        positionsC[i] = (matC['coordinates'][i][0],matC['coordinates'][i][1])
    GC = nx.from_numpy_matrix(sparse.csc_matrix(matC['A']).todense(), parallel_edges = False)
    nx.set_node_attributes(GC, positionsC, 'coord')
    posC = matC['coordinates']
    #normalize edge weights for viweing
    StandardEdgeWeightsC = standardWidth(GC) + 0.5
    print("Conductance")
    nx.draw(GC, pos =posC, width = StandardEdgeWeightsC, node_size = 0.2)
    plt.show()

    matP = scipy.io.loadmat('fungal_MATLAB\PathScore' +filename )
    positionsP = dict()
    for i in range(len(matP['coordinates'])):
        positionsP[i] = (matP['coordinates'][i][0],matP['coordinates'][i][1])
    GP = nx.from_numpy_matrix(sparse.csc_matrix(matP['A']).todense(), parallel_edges = False)
    nx.set_node_attributes(GP, positionsP, 'coord')
    posP = matP['coordinates']
    #normalize edge weights for viweing
    StandardEdgeWeightsP = standardWidth(GP) + 0.5
    print("Path Score:", )
    nx.draw(GP, pos =posP, width= StandardEdgeWeightsP, node_size = 0.2)
    plt.show()
```

```python
#############THIS IS THE MAIN WORKING SIMULATION#####(1/2)

# #initialize graph
Gsim = nx.Graph()
HarvestRate = 7 #ADJUST VALUE
nx.set_node_attributes(Gsim, 0 , 'food')
nx.set_node_attributes(Gsim, (0,0), 'coord')

nx.set_edge_attributes(Gsim, 0, 'flow')
nx.set_edge_attributes(Gsim, 0, 'resistance')
nx.set_edge_attributes(Gsim, 0, 'weight')
nx.set_edge_attributes(Gsim, 0, 'length')#technically dont need can be sourced from node data
nx.set_edge_attributes(Gsim, 0, '')

#set food sources map
f1 = Polygon([(0,0),(0,0.3),(0.3,0.3),(0.3,0)])
f2 = Polygon([(5,5),(5,6),(6,5)])
f3 = Polygon([(2,3),(2,2),(3,3)])
foodMap = MultiPolygon([f1,f2,f3])
#intial innoculation points
Gsim.add_node(0, food = HarvestRate, coord=(0.2,0.2))
Gsim.add_node(1, food = HarvestRate, coord=(5.2,5.2))
nodeNum = 1
# Gsim.add_node(2, food = HarvestRate, coord=(0.5,5.3))
# nodeNum = 2


#node iteration
tempGN = list(Gsim.nodes(data =True))#holds static for loop
for Nodes in tempGN:
    if( hasFood(Nodes[1]['coord'], foodMap)):
        Nodes[1]['food'] = HarvestRate
        #distribute food to immediately connected edges or add new edge
        #hyphal growth or distribute to edges??

        while(Nodes[1]['food'] > 0):#distribute all harvested food
            #coordinates, weight, length, food left
            NewGrowth = hyphalGrowth(Nodes[1]['coord'], Nodes[1]['food'] )
            nodeNum = nodeNum +1
            Gsim.add_node(nodeNum, food = 0, coord= NewGrowth[0])
            Gsim.add_edge(Nodes[0],nodeNum, weight = NewGrowth[1], length = NewGrowth[2], Rigidness = 0, PhaseRigid = 0, PhaseAbsFlow = 0, PrevPhFlow = NewGrowth[1]*NewGrowth[2])
            Nodes[1]['food'] = NewGrowth[3]

tempGE = list(Gsim.edges(data =True))
plotSim(Gsim, foodMap)
```

```
#############THIS IS THE MAIN WORKING SIMULATION#####(2/2)
####Executes single phase iteration for nodes and edges
##Repeat execution to run more phases
#decreasing harvest rate with fluctuation
HarvestRate = HarvestRate/2
if(HarvestRate < 0.2):
    HarvestRate = ranged_normal(0.2, 0.1, 0, 0.5)[0]
# iterate through nodes
tempGN = list(Gsim.nodes(data =True))#holds static for loop
for Nodes in tempGN:
    if( hasFood(Nodes[1]['coord'], foodMap)):#nodes only distribute resources if they are located over food sources
        Nodes[1]['food'] = HarvestRate
        #distribute food to immediately connected edges or add new edge
        #hyphal growth or distribute to edges??
        neighborWsum = sum([x[2] for x in list(Gsim.edges(Nodes[0], data = 'weight'))])
        hyphalFood = ranged_normal(0.2, 0.1, 0, 1)[0] * Nodes[1]['food']#ADJUST VALUE controls resource distribution to new edges or existing edges
        contFood = Nodes[1]['food'] - hyphalFood
        for kEdge in Gsim.edges(Nodes[0]):
            contFood = porpEdgeGrowth(Gsim, kEdge,  contFood, neighborWsum)

        Nodes[1]['food'] = hyphalFood
        while(Nodes[1]['food'] > 0.1):#ADJUST VALUE ,leftovers
            #coordinates, weight, length, food left
            NewGrowth = hyphalGrowth(Nodes[1]['coord'], Nodes[1]['food'] )
            nodeNum = nodeNum+1
            Gsim.add_node(nodeNum, food = 0, coord= NewGrowth[0])
            Gsim.add_edge(Nodes[0],nodeNum, weight = NewGrowth[1], length = NewGrowth[2], Rigidness = 0, PhaseRigid = 0, PhaseAbsFlow = 0, PrevPhFlow = NewGrowth[1]*NewGrowth[2])
            Nodes[1]['food'] = NewGrowth[3]


#iterate through edges
tempGE = list(Gsim.edges())
for Edgek in tempGE:
    #total resources capable of distributing
    transfer = ResourceTransferAmount(Gsim, Edgek)
    #TODO pressure causes hyphal growth vs. edge growth
    hyphalFood = ranged_normal(0.2, 0.05, 0, 0.4)[0] * transfer #add paramaters to distribution to affect hyphal growth/decay
    contFood = transfer - hyphalFood
    AllEdges = list(Gsim.edges(Edgek[0])) #TODO need to make asyncronous
    AllEdges += list(Gsim.edges(Edgek[1]))
    neighborWsum = sum([Gsim.edges[x]['weight'] for x in AllEdges] ) #TODO make asychronous
    NWS = sum([x[2] for x in list(Gsim.edges(Edgek[0], data = 'weight'))]) + sum([x[2] for x in list(Gsim.edges(Edgek[1], data = 'weight'))])
    for kEdge in AllEdges :
            contFood = porpEdgeGrowth(Gsim, kEdge,  contFood, NWS)
    i = 0
    while(hyphalFood > 0.01):####VALUE TO ADJUST
        if(i == 0):#switch between which side the resources are being transfered to
            i = 1
        else:
            i = 0
        #TODO hyphal growth from edge midpoints instead

        #coordinates, weight, length, food left
        NewGrowth = hyphalGrowth(Gsim.nodes[Edgek[i]]['coord'], hyphalFood )
        nodeNum = nodeNum +1
        Gsim.add_node(nodeNum, food = 0, coord= NewGrowth[0])
        Gsim.add_edge(Edgek[i],nodeNum, weight = NewGrowth[1], length = NewGrowth[2], Rigidness = 0, PhaseRigid = 0, PhaseAbsFlow = 0, PrevPhFlow = NewGrowth[1]*NewGrowth[2])
        hyphalFood = NewGrowth[3]
```

```
#remove small weighted edges
for Edgek in  list(Gsim.edges()):
    if(Gsim.edges[Edgek]['weight'] < 0.01):####VALUE TO ADJUST BETWEEN ITERATIONS

        Gsim.remove_edge(Edgek[0], Edgek[1])

Gsim.remove_nodes_from(list(nx.isolates(Gsim)))
nodeNum = ResolveIntersections(Gsim, nodeNum)#takes a long time to run but is neccessarry


plotSim(Gsim, foodMap)
```