PNAS
www.pnas.org

Supplementary Information for

Cooking Beyond Your Front Door: Novel Fusion Recipe Generation

Tiarnán Finn-James Curran-Feeney

Tiarnán Finn-James Curran-Feeney.
E-mail: tiarnan.curran-feeney@warwick.ac.uk

This PDF file includes:

Tables S1 to S2

1. CulinaryDB: A Critical Eye

CulinaryDB contains 45,772 recipes comprised of 1,033 ingredients from 22 defined regions globally taken from multiple websites. Ignoring controversial region names, such as Britain and Ireland being classified under the outdated label of British Isles, there are downfalls to the data set. Firstly, the data set restricts itself to English speaking sites which leads to a better representation of the USA and western Europe than the rest of the world. Secondly, due to the large list of ingredients that had to be classified there is the occasional wrong or debated classification of an ingredient. A collection of such classifications are lised in Table S1.

Thirdly, in order to simplify the data's complexity, the ingredients in recipes are reduced to be contained in one of the 1,033 ingredients previously mentioned. This allows for easier to run code but has resulted in some unusual decisions. Included in Table S2, we see a sample of unusual choices in regards the labelling of ingredients. Many oils, such as olive oil, are relabelled by what they are made from, like olives. While this reduces the pool of possible ingredients, even the most professional chef would be stumped trying to deep fry doughnuts in a vat of sunflower seeds.

When classifying recipes by region there is no explanation as to how this was done, leading to question whether the region classification can be relied upon fully. Especially with websites such as AllRecipes.co.uk, where anyone can post a recipe.

Table S1. Sample of unexpected ingredient classifications.

File	Row	Ingredient Name	Classification
02_Ingredients	333	Oregano	Spice
02_Ingredients	424	Mexican Oregano	Herb
03_Compound_Ingredients	58	Ranch Dressing	Spice

Table S2. Sample of unexpected ingredient labels contained within 04_Recipe-Ingredients_Aliases.csv.

Row	Original Ingredient	Labelled
17138	Olive Oil	Olive
5808	Sesame Oil	Sesame
26726	Taco Seasoning	Taco
56824	Burrito Seasoning	Burrito

2. Project Code

```python
#Loading in required modules
import pandas as pd
import numpy as np
import networkx as nx
from matplotlib import pyplot as plt
import math

#Load in CulinaryDB database
recipeDetails = pd.read_csv(filepath_or_buffer='./Data/01_Recipe_Details.csv', sep=',', encoding='latin'
    )
ing = pd.read_csv(filepath_or_buffer='./Data/02_Ingredients.csv', sep=',', encoding='latin')
compIng = pd.read_csv(filepath_or_buffer='./Data/03_Compound_Ingredients.csv', sep=',', encoding='latin')
recipeIng = pd.read_csv(filepath_or_buffer='./Data/04_Recipe-Ingredients_Aliases.csv', sep=',',
    encoding='latin')

#Record number of ingredients (compound and simple)
ingShape = ing.shape
compIngShape = compIng.shape
totIng = ingShape[0] + compIngShape[0]

#Record Number of Regions
regionList = np.unique(recipeDetails['Cuisine'].values.tolist())
regions = regionList.shape[0]

#Quick map from region name to layer of matrix
areaCode = {}
for i in range(regions):
    areaCode[regionList[i]] = i

#Quick map from ingredient code to node number
ingCode = {}
ingredients = ing['Entity ID']
for i in range(ingShape[0]):
    ingCode[ingredients[i]] = i
ingredientsComp = compIng['entity_id']
for i in range(compIngShape[0]):
    ingCode[ingredientsComp[i]] = ingShape[0] + i

#Quick map from node number to ingredient name
labels = {}
for j in range(totIng):
    if (j < 930):
        labels[j] = ing['Aliased Ingredient Name'][j]
    else:
        labels[j] = compIng['Compound Ingredient Name'][j-930]

#Create adjacency matrix for each region
adjMx = []
for i in range(regions):
    adjMx.append(np.zeros([totIng,totIng]))

#Store dist of ingredient list length
ingDist = []
for i in range(regions):
    ingDist.append([])

#Record number of recipes and length of ingredient file
fileLength = recipeIng.shape[0]
recipeCount = recipeDetails.shape[0]
startPoint = 0;

#Iterate through each recipe
for i in range(recipeCount):
    #Find which ingredients are in the recipe
    ingredientList = []
```

```
    ingLength = 0;
    while (recipeIng.iloc[startPoint,0] == i+1):
        #Add ingredient to ingredient list and update
     number of ingredients
        ingredientList.append(ingCode[recipeIng.iloc[
     startPoint,3]])
        startPoint = startPoint + 1
        ingLength = ingLength + 1
        #Stop once we are at a new recipe
        if (startPoint == fileLength):
            break
    #Check region
    region = areaCode[recipeDetails.iloc[i,3]]

    #Add recipe ingredient list length to distribution
    ingDist[region].append(ingLength)

    #For each ingredient add one to frequency of
     occurrence together in adj mx
    count = len(ingredientList)
    for j in range(count):
        for k in range(count):
            adjMx[region][ingredientList[j]][
     ingredientList[k]] += 1

G = [] #List of graphs

#Create Weights
for i in range(regions):
    #Remove all self-loops
    for j in range(totIng):
        adjMx[i][j][j] = 0

    #Create Weighted graph
    G.append(nx.from_numpy_matrix(adjMx[i]))

#Save Properly for MATLAB WSBM code
for i in range(len(G)):
    A = nx.adjacency_matrix(G[i])
    np.savetxt(".\\adjMx\\adjMx" + str(i) + ".txt", A.
     todense(),delimiter= " ")
```

Listing 1. Constructing network and storing useful information.

```
function [labels,model] = saveAdjMx(regions,alpha,beta)

    %Construct adjacency matrix
    number = length(regions);
    adjMx = zeros(1033*number);
    for i = 1:number
        for j=1:number
            adjMx((1 + 1033*(i-1)):(1033*(i)),(1 +
     1033*(j-1)):(1033*(j))) = beta .* eye(1033);
        end
    end

    %Load in matrix from saved files
    for i = 1:number
        newMx = readmatrix(strcat('..\adjMx\adjMx',
     string(regions(i)),'.txt'));
        adjMx((1 + 1033*(i-1)):(1033*(i)),(1 + 1033*(i
     -1)):(1033*(i))) = newMx;
    end

    %Set non-edges to "NaN"
    adjMx(adjMx == 0) = nan;
    %Subtract one from each edge weight
    adjMx = adjMx - 1;

    %Run WSBM
    [labels,model] = wsbm(adjMx ,10,'W_Distr','Poisson'
     ,'alpha',alpha,'numTrials',12,'parallel',1);

    %Save data
    thetaW = reshape(model.Para.tau_w(:,1),10,10);
    thetaE = reshape(model.Para.theta_e,10,10);

    writematrix(thetaW, strcat('..\params\thetaW',
     string(beta),'.txt'))
    writematrix(thetaE, strcat('..\params\thetaE',
     string(beta),'.txt'))
    writematrix(labels,strcat('..\params\z',string(beta
     ),'.txt'))
    writematrix(regions,strcat('..\params\regions',
     string(beta),'.txt'))
end
```

Listing 2. MATLAB function to construct multilayer network and run WSBM model.

```
#Construct new graph from MATLAB data
totIng = 1033
Gnew = []

#Load in blockstructure data
thetaE = np.loadtxt(".\params\\thetaE10.txt",delimiter
     = ",")
thetaW = np.loadtxt(".\params\\thetaW10.txt", delimiter
      = ",")
z = np.loadtxt(".\params\\z10.txt",dtype =int,
     delimiter = ",")
newRegionList = np.loadtxt(".\params\\regions10.txt",
     dtype =int, delimiter = ",")

#Generate each layer
for k in range(int(len(z) / totIng)):
    Gnew.append(nx.Graph())
    Gnew[k].add_nodes_from(range(totIng))
    for i in range(totIng):
        for j in range(i+1,totIng):
            edge = np.random.random_sample()
            if (edge < thetaE[z[i]-1][z[j]-1]):
                Gnew[k].add_edge(i,j,weight = np.random
     .poisson(thetaW[z[i]-1][z[j]-1]) + 1)
```

Listing 3. Generating new network using block structure information.

```
#Pick layer or "region"
region = 0

#collect nodes by their group
group = [[],[],[],[],[],[],[],[],[],[]]
for i in range(1033):
    group[z[i+1033*region]-1].append(i)
blockStructure = []

#Construct order for nodes based on block structure
for g in group:
    blockStructure = blockStructure + g

#List of all unused nodes
remove = [node for node,degree in dict(G[newRegionList[
     region]].degree()).items() if degree == 0]

#Default order
defaultStructure = list(range(1033))

#Remove unused nodes from both orders
for i in remove:
    blockStructure.remove(i)
    defaultStructure.remove(i)

#Load both adjacency matrices
adjNew = nx.adjacency_matrix(G[newRegionList[region]],
     nodelist=blockStructure,weight='weight')
adjOld = nx.adjacency_matrix(G[newRegionList[region]],
     nodelist=defaultStructure,weight='weight')

#Plot results
fig = plt.figure(figsize=(20,40))

ax1 = fig.add_subplot(121)
ax1.imshow(adjOld.todense(), cmap="copper_r")

ax2 = fig.add_subplot(122)
ax2.imshow(adjNew.todense(), cmap="copper_r")

plt.savefig('block' + str(region) + '10.png',
     bbox_inches='tight')
plt.show()
```

Listing 4. Python code to display the adjacency matrix for each layer.

```
#Modified Random Walk Proposed
def modRdmWalkPort(G,region,x0,maxIng,gamma,delta):
    #x is an ingredient list
    x = [x0]
```

```
    #storing info on nerby ingredients as well as
    weights of edges for each ingredient in list x
    ing = [[] for _ in range(len(G))]
    weight = [[] for _ in range(len(G))]
    total = [[] for _ in range(len(G))]
    invWeight = [[] for _ in range(len(G))]

    #Adding ingredients to our list
    for i in range(maxIng-1):

        #Deciding whether to choose randomly, change
    region or just pick an ingredient
        choice = np.random.choice([0,1,2],p=[gamma,
    delta, 1-gamma-delta])

        #For the new ingredient added we have to store
    the information found
        for j in range(len(G)):
            #Getting adjacent ingredient and edge
    weight info for region j
            nextIng = []
            nextWeight = []
            for u,v,w in G[j].edges(x[i],data=True):
                nextWeight.append(w['weight'])
                nextIng.append(v)

            #Storing all information
            total[j].append(sum(nextWeight))
            ing[j].append(nextIng)
            weight[j].append(nextWeight)

            #Storing inverted total weight (making sure
     to account for when the weight sums to zero)
            if (total[j][i] < pow(10,-6)):
                invWeight[j].append(0)
            else:
                invWeight[j].append(1/total[j][i])

        #Updating region (if applicable)
        if choice == 1:
            newRegion = np.random.choice(range(len(G)
    -1))
            if newRegion < region:
                region = newRegion
            else:
                region = newRegion +1

        if choice == 0:
            #Choosing random ingredient if applicable
            x.append(np.random.choice(G[region].nodes()
    ))
        else:
            #Summing all inverted weights
            totInvWeight = sum(invWeight[region])
            if totInvWeight == 0:
                x.append(np.random.choice(G[region].
    nodes()))
            else:
                #Picking an ingredient from list x with
     probability proportional to inverse of
                #sum of weights on edges leaving it
                prob = [w/totInvWeight for w in
    invWeight[region]]
                j = np.random.choice(range(len(x)),p=
    prob)

                #Picking ingredient neighbouring our
    chosen ingredient from list x
                newIng = np.random.choice(ing[region][j
    ],p=[w/total[region][j] for w in weight[region][j
    ]])
                x.append(newIng)

    return x

#Random Walk for a single layer
def rdmWalkPort(G,x0,maxIng,gamma):
    return modRdmWalkPort([G],0,x0,maxIng,gamma,0)
```

Listing 5. Python function to perform the modified random walk proposed in **Recipe Generation Strategy, Materials and Methods**.

```
#Function to run through layer G of the graph for
    recipes of length L for a chosen gamma
def recipeLenDist(G,L,iterNo,gamma):
    lengthDist = []
    for j in range(iterNo):
        x = rdmWalkPort(G,np.random.choice(G.
    number_of_nodes()),L,gamma)
        #if len(set(x)) > 1:
        lengthDist.append(len(set(x)))
    return lengthDist

#Variables to change in order to match histogram to
    preferred region
gamma = 0.1
length = 30
region = 5
#Run function
lengthDist = recipeLenDist(Gnew[region],length,10000,
    gamma)
#Plot results
plt.hist(ingDist[newRegionList[region]],len(set(ingDist
    [newRegionList[region]])), density=True, facecolor
    ='b', alpha=0.5,label='True Distribution')
plt.hist(lengthDist, len(set(lengthDist)), density=True
    , facecolor='r', alpha=0.5,label='Distribution')
plt.xlabel('Recipe Ingredient List Length', fontsize
    =16)
plt.ylabel('Probability', fontsize=16)
plt.title('Historgram showing the Distribution of
    Number of\nUnique Nodes Visited over 10,000 Random
     Walks\nof length ' + str(length) + ' with $\
    gamma$ =' + str(gamma), fontsize=20)
plt.legend(prop={"size":16})
plt.savefig('..\Plots\lenDist' + str(region) + '_' +
    str(gamma) + '_' + str(length) + '.png',
    bbox_inches='tight')
plt.show()
```

Listing 6. Python code to generate recipe ingredient list length distribution for modified random walk.

