A Food Network
A Graphical Exploration of Food Recipes

William MacDonald Carr

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Masters of Science

in Data Science

July 2022

APPROVED:

____

Professor Yanhua Li, Major Thesis Advisor

Contents

Abstract

This paper presents several approaches of converting food recipe data into graph data structures. We attempt to model these recipes using a directed rooted tree graph network. With this data structure we are able to give a standardized structure to food recipes - which in turn makes them tractable to a variety of machine learning algorithms. Previous attempts at analyzing food recipe data have viewed this problem fundamentally as a Natural Language Processing problem - this view however does not take advantage of the inherent structure already within food recipes. The goal of this paper is to begin the exploration of potential graph architectures which can be applied to food recipes - and hopefully extrapolated to processes in other domains which lend themselves easily modelable by these data structures.

Terms: Graphs,Data Structures,Machine Learning, Culinary

Acknowledgements

I would like to thank my wife, Helen, for being so helpful and understanding while I chipped away at this project. I also want to thank my baby Claire - for patiently waiting for me while I spent countless hours at my computer.

Background & Motivation

Food is life. We all eat everyday in order to survive - and recipes are the directions to put together these creations. As such most people have inherent expert level intuition as to which food ingredients pair well - e.g bread and butter and not MSG and cod - and yet to our knowledge there is yet no model built specifically for food recipe data. Additionally, there is no standard data structure by which to encode recipe data nor is there a single repository for food recipes - e.g. *Wikipedia* of Recipes. This paper seeks to open the door to the exploration, curation and study of food recipes from a standardized format.

Food is in many ways analogous to human language - both are simultaneously vessels and integral components of culture. Just as languages evolve over time, so too do cuisines. Ingredients are borrowed; techniques are passed down and flavor pairings melded - but on the other hand countless recipes are lost to the ages merely because no one is transcribing them. These ingredients and recipes reveal the daily lives of individuals as well as broad historical trends. Food gives another lens through which to view and study human migration, culture, religion as well as non-human factors such as weather, climate, earthly phenomena (e.g. volcanoes, droughts, etc.) and even cosmological events.

The original motivation for this project stemmed from personal experience dealing with numerous online recipes. Each website has its own layout, presentation and set of instructions. In general the recipes (instructions) are buried between walls of text which oftentimes are irrelevant to the recipe. There are many websites though, which display simplistic and straightforward recipes but these websites are disparate and disconnected. They often tend to be cuisine specific and small in scope *(e.g. online Amish cooking recipes, Chinese vlogger recipes,etc.)* A common thread throughout all these sources is the lack of standardization. Especially when it comes to how to break down a set of instructions into its component steps. The exact same recipe could be three steps on one website, while five on another.

This realization that there was apparently no standardized way of describing the preparation of a dish led me to ponder on ways such a process could be modeled. This lack of standardization is understandable as there is an incredible level of complexity and nuance in the sourcing, preparation and ultimate construction of a recipe. Thus attempting to apply the cold, calculating and precise approaches prescribed by mathematics to food seems akin to mixing olive oil and vinegar - pun intended. Nevertheless this does not mean that this process can not be modeled. A model ipso facto can never perfectly match reality, but if it can give us a better understanding and perspective into phenomena we are trying to study then it is a useful* model which can inform further exploration and inquiry.

Another motivation behind standardizing food recipes is that it would facilitate the creation of an online repository for food recipes. Data stored in this repository would need to adhere to certain standards/data structures - not solely for conformity's sake but rather to allow for easier exploration and analysis both for human users and computer algorithms.
We envisage this as something similar to Wikipedia whereby anyone can create and submit a recipe which can be tagged, ranked and reviewed by others. User submitted data can be used to further train models. Ultimately, our hope is that with sufficient data, appropriate feature labeling and a representative model we can use machine learning algorithms to generate realistic synthetic food recipes. These food recipes could be generated according to conditional

inputs/restraints based on the user. For example, Chinese recipes with avocado and bacon - these may not exist so the model should try to generate some recipe which has both bacon and avocado but also has elements which make it similar to Chinese food.

When this is tied together with other systems we could create some very interesting applications. For example a phone application that can give you recipes based on the food in an image (e.g. in the refrigerator). We also imagine a program that could determine which recipes to use such that it optimizes some metric - e.g. minimize waste, maximize nutrition, minimize time, etc. This could be impactful in large feeding situations - such as mess halls, homeless shelters, refugee camps, etc.

Problem Statement

As mentioned in the Motivation section of this document, one of the purposes of this research is to open the door to the exploration and mapping of food recipes in a standardized format. The hope of this is that we will then be able to use this data to analyze recipes from a data-centric perspective.

As an initial step toward this direction, we present a graph structure for food recipes. Using this architecture we wish to transcribe and convert said recipes into a format where we can apply machine learning techniques to predict whether a given recipe is Real or Fake (synthetic). More specifically, we wish to train a MLP Classifier on graphical data of the form: **G(V,E,u)**, (vertices, edges, universal features) such that it will be able to distinguish Real Recipes from Synthetic (Fake) Recipes.

This data structure, which we will further expound upon later in the paper, uses directed tree graphs to model the recipes. We also propose a hypergraph structure to model recipes. This data structure has not been fleshed out in as much detail as its lower-dimensional analog - but we suspect this data structure should allow for us to model and describe complex interactions between multiple entities more fully and accurately. This idea will be explored towards the end of our paper. Additional data structures such as dynamic graphs, flow networks, etc. were explored and should still be considered as candidate data structures for modeling food recipes.

Graph Data

Graph data is notoriously difficult to work with but at the same time it is one of the most universal and fundamental data types. At their most basic, graphs simply represent relations between entities. This flexibility allows graphs to model a wide variety of relations and phenomena. Graph data is commonly encountered in network analysis, social networks, chemistry, knowledge graphs and even image recognition - images can be viewed as a lattice of nodes. (*See Figure 1)*.

Graph data's complexity lies in the fact that it has very few assumptions regarding the structure of the data *(See Figure 2)*. Data used for most machine learning models is standardized in its input size (e.g. MNIST 28 x 28) or if not, can often be made so via padding, etc. Getting a standardized representation for graphs, however, is a much thornier issue. Not only is the structure of graphs dependent on *both* nodes and edges, but nodes and edges can each contain feature information as well. In order to obtain a standardized representation we can employ a variety embedding techniques

Dataset

The data used in this project was pulled from a variety of food recipe websites online. In the original scope of the project, it was intended to make full use of a dataset of ~18,000 recipes directly from *FoodNetwork.com*. Ultimately however, due to the difficulties of parsing each recipe fully and accurately we instead relied on a manual process of transcribing recipes. These recipes largely consisted of sauces, rubs and other recipes which themselves are often used in cooking (e.g. chicken stock). Attention was made to include a variety of recipes from different regions/cultures - that being said, there is a bias towards more Western orientated recipes.

Our data is organized in a graphical structure - where each recipe corresponds to a directed rooted tree graph **G(V,E,u)**. Each node $\mathbf{v}_i \in \mathbf{V}$ represents an ingredient used in the recipe - e.g. *broccoli*, *salt*, etc. Each edge, $\mathbf{e}_i \in \mathbf{E}$**,** corresponds to a sequence of actions performed on an ingredient or collection of ingredients. The graph label, **u**, contains feature information about the graph as a whole - in our case **u** represent whether a recipe is real or fake - represented by **0** or **1** respectively. We will further define these concepts below.

Nodes

Under our construction, every node is one of two types - base ingredient node or aggregation node. Base Ingredient nodes denoted by $\mathcal{I}$ - are defined as a *pure,*unadulterated ingredient which itself is not a recipe. For example, *broccoli,corn,beef,water,tea* and *pumpkin seeds*, are all base ingredients - while *ketchup, soy sauce* and *ice* are what we call *functional ingredients*. In order to incorporate these *functional ingredients* into our recipes, we developed an iterative process which *deconstructs* a *functional ingredient* into its component graph which is then woven back into the original graph structure.

Since most recipes involve *functional ingredients*, almost all graphs are generated using this *deconstruction* process. Many *functional ingredients* require multiple iterations to get to their base ingredient form - e.g. ***Heinz 57 Sauce*** is made with ***ketchup*** which in turn is made with ***vinegar*** - (and that in turn is made from a vinegar culture and alcohol). The reason for this tedious and exhaustive process is so that all component nodes can exist in the same node feature space - this is expounded upon below in the Node Features section.

More specifically, Under our construction, the set **V** of nodes can be subdivided into two types - namely base ingredients and aggregations - denoted by $\mathcal{I}$ and **Σ**,. Note $|\mathbf{\Sigma}| = |\mathcal{I}| = \mathbf{n}$, where n is the dimension of the feature space for the nodes . All aggregation nodes, by definition exist to aggregate information on their neighbors and do not contain any inherent feature information themselves, hence all elements of $\mathbf{\Sigma} = \{ \mathbf{0}^n \}$ - n-dimensional zero vector.

Node Features

The features of the ingredient nodes are a concatenation of four separate ingredient features- namely the ingredients' relative location on an extended taxonomic tree; the structure/form of the ingredient; the ingredients flavor profile and its age.

The extended taxonomic tree of life corresponds to each ingredient's position on an extended biological tree of life. We hope to capture the idea of similarity or distance between various nodes in the tree of life. In this way ,for example, we can quantify the idea that a jellyfish is more similar to a squid than it is to a potato. The structural component corresponds to the form which the ingredient takes on. This is used to distinguish between roots, flowers, stems, bones, meat et cetera. This is also similarly organized in a hierarchical tree structure (albeit smaller) like the tree of life. For example if we were to compare a pumpkin vs. pumpkin seeds - under this construction both ingredients' taxonomic features would be the same, however their structural/form features would be different (as well as flavor profile). The final features, flavor profile and age are comparatively simple - flavor profile is a one hot encoded vector representing the presence (or absence) of various flavors - and age is an ordinal variable (can be treated as continuous) representing whether an ingredient is *young, mature*, etc.

Faithfully converting these features in a coherent and useful manner necessitated exploring a variety of embedding techniques.Specifically, one of the difficulties faced here is how to represent the idea of distance on a graph structure. Originally given that our tree of life was in a graph structure we decided to employ various graph walking techniques to try to approximate the topology of the graph structure. However these approaches faced significant issues to approximating the structure due to both the graph's size and the fact that the graph walking algorithms do not take explicit advantage of the hierarchical structure of the data. For example the distance from animal to fungus is a larger distance than the distance from cat to dog -since more nodes need to be hopped en-route from cat to dog, the calculated distance from cat to dog will be larger - i.e. **d**(*cat,dog*) > **d**(*animal,fungus*).

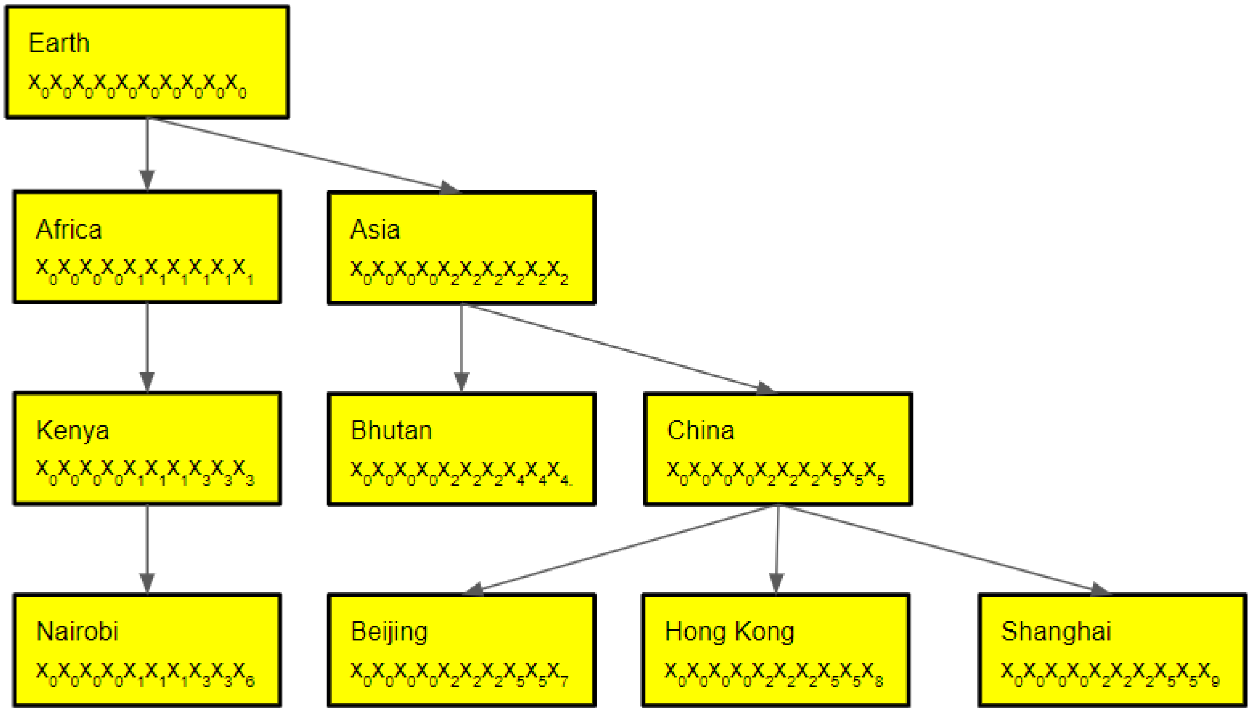
Animal to Fungus: *animal→biotic→fungus*
Cat to Dog: *cat→felis→felidae→carnivora→ canidae→canis→dog*

Various techniques can be employed to help ameliorate this problem - such as additional connections (e.g. between cousin nodes), weighted walks, etc. Tweaking these algorithms may have led to more acceptable embeddings, however we felt that continuing down this path was akin to trying to fit a square peg in a round whole. Rather than artificially massage the data into an acceptable form, we sought out other embedding techniques which take advantage of the hierarchical structure of the data.

Hierarchical Feature Algorithm

Given that these features have an inherent biological, hierarchical nature we decided to try to incorporate concepts from genetics to approximate the idea of distance between the elements (nodes) of the graph. In our loose approximation of genetic biology, we assume a child node inherits its parent node's (each node has one and only one parent node) genetic code (represented by a string) and that it has some change to its code, making it different from its parent. The amount of change is proportional to the hierarchical level of the parent/child node. In this way we can get a representation which encapsulates the idea of genetic distance - the distance between two nodes.

To get a more concrete understanding of what is going on, we present a toy example below:


Earth
$x_0x_0x_0x_0x_0x_0x_0x_0x_0x_0$
Africa
$x_0x_0x_0x_0x_1x_1x_1x_1x_1x_1$
Asia
$x_0x_0x_0x_0x_2x_2x_2x_2x_2x_2$
Kenya
$x_0x_0x_0x_0x_1x_1x_1x_3x_3x_3$
Bhutan
$x_0x_0x_0x_0x_2x_2x_2x_4x_4x_{4.}$
China
$x_0x_0x_0x_0x_2x_2x_2x_5x_5x_5$
Nairobi
$x_0x_0x_0x_0x_1x_1x_1x_3x_3x_6$
Beijing
$x_0x_0x_0x_0x_2x_2x_2x_5x_5x_7$
Hong Kong
$x_0x_0x_0x_0x_2x_2x_2x_5x_5x_8$
Shanghai
$x_0x_0x_0x_0x_2x_2x_2x_5x_5x_9$

Node	Earth	Africa	Asia	Kenya	Bhutan	China	Nairobi	Beijing	Hong Kon0	Shanghai
Index	$\mathbf{x_0}$	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{x_4}$	$\mathbf{x_5}$	$\mathbf{x_6}$	$\mathbf{x_7}$	$\mathbf{x_8}$	$\mathbf{x_9}$
Level	3	2	2	1	1	1	0	0	0	0
#Char	4	3	3	2	2	2	1	1	1	1

Here the #Char is simply defined by the relation ; $f_{\#Char}(\text{Level}) = 1 + \text{Level}$.

More generally wecan define $f_{\#Char}(\text{Level})$ as any strictly increasing function satisfying the following properties:

1. $f_{\#Char}(\text{Level\_a}) \geq f_{\#Char}(\text{Level\_b})$ iff Level_a $\geq$ Level_b.
2. $f_{\#Char}(\text{Level}) \in \mathbf{Z^+}$

Examples of such functions include *c* (constant), $\lfloor log(x) \rfloor$, $\lfloor x^a \rfloor$ where $a \geq 0$.

We have chosen our $f_{\#Char}(\text{Level}) = \lceil 1.5^{Level} \rceil$, our intuition is that there is an increasing difference between nodes in each successive hierarchical level, we keep the base low - *1.5* - so as to give sufficient weight to parent nodes, while still allowing nodes occupying lower levels on tree to make an impact as well.

Given a root node, R of a tree T whose nodes are ordered, we assign strings to each node as follows:

For each node in G, assign a unique number 0,...k-1, where k = |G| - we will represent the root node with x_0 by convention, i.e. number(R) = 0.

Algorithm:

Total String Length = $\sum_{level=0}^{Max\,Tree\,Depth} f_{\#Char}(level)$

Remaining Length = $\sum_{0}^{Depth} f_{\#Char}(level)$

```
for node in Tree:
    string = ‘’
    level = Level(node)
    node_index = index(node)
    child_node = node.child
    if child_node:
        characters = $x_{node\_index} * \sum_{0}^{level} f_{\#Char}(level)$
    else:
        characters = $x_{node\_index} * f_{\#Char}(level)$
    string = characters
    parent_node = node.parent
    while parent_node:
        level += 1
        node_index = index(parent_node)
        characters = $x_{node\_index} * f_{\#Char}$(level)
`       string += characters
        parent_node = parent_node.parent
    return string
```

The approach was as follows, first establish the graph structure of the tree of life *(See Figure 3)* - we did this by merging the lineages of every ingredient used in the recipe data set. Note that this tree of life is not in 100% concordance with the Tree of Life in biology. The first major difference is the size - our tree is deeper (Starting from Non-Life ending in Sub varieties) though much less broad. Our tree's increased depth is to account for all the things we used as ingredients (e.g. salt, MSG, varieties of peppers, etc.). As for its breadth, since our tree was constructed from its constituent pieces (lineages), any ingredient which does not show up in our data will not be incorporated (sorry humans, anteaters, poison ivy, etc.) in the process.

Other techniques and algorithms do exist which give a much more accurate and full representation of the data, but for our purposes, due to the limitations of computing and DNA records we use the aforementioned.

After creating this hierarchical tree structure data structure - based on the taxonomic tree of life - we draw further inspiration from the field of genetics by giving a string representation to each node on said tree - more on these later.

Our basic idea is to use a sequence of letters or characters to represent the genetics of a given point on the node. At each subsequent sub level in the hierarchy we assume there is a smaller contribution to this "genetic code". There exist 11 rungs in our extended taxonomic tree. As noted above we assumed that each successively higher level makes a larger contribution to the *genetic code* for a given node.

We start by using one character to represent the difference between subvarieties and subsequently increase the number of characters added by 50% per hierarchical level (rounded up) of the number of additional/changes we make to the genetic code. Each node has its own unique character which contributes to the *genetic code* of not only itself, but also for any nodes which descend from that node. *(Refer to the toy example above for a more visual explanation of how the process works)*. Our original most basic node "everything" - the root node - contains 160 characters (all x_0). As we continue to traverse the graph starting from the root node, Each node makes its own contribution proportional to its position in the hierarchy. Note that each node in the hierarchical tree of life may or may not be used in the data set. For example the biological order *lago morpha* is not used in the dataset however rabbits and hares which belong to that order *are* used- one reason why this is important is that certain recipes call for generic ingredients which can not be easily represented by a single species (e.g. recipes calling for fish, meat, berries, etc.).

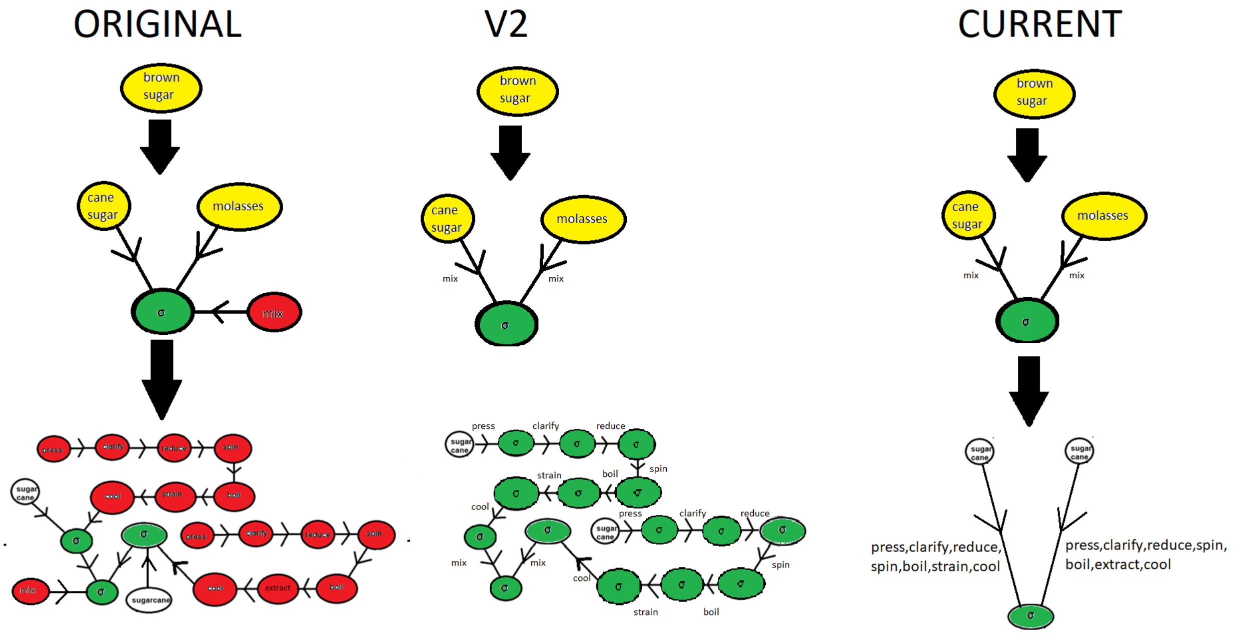
After assigning each node a 160 character string we can now begin to measure distance between the various nodes (strings). To compare the distance between the strings to one another we use a Hammond distance or edit distance metric. After computing the distances between each pair of nodes, we can now begin to place these nodes in a metric space relative to another. However, since the dimension of this feature space is so large we wish to further transform the data into a less sparse feature space. We wish to reduce the size of this feature space in a way that preserves both the global and local structure of the data. A variety of dimensional techniques exist, but we ultimately chose to use UMAP as it not only preserves a good balance of both local and global structures but it is also compatible with a Hammond distance metric. Using this we reduced the size of the feature space from ~ 800 to 11. We settled on 11-dimensional projection after testing other projection techniques and finding that a 11 dimensional PCA representation explained 95% of the variance (as when compared to the original) - unfortunately since UMAP relies on more topological techniques it cannot be compared in the same manner. Even when projecting this to 2 dimensions *(See Figure 4)* we can see that the projection does an excellent job at preserving the structure of the data - the differences between the groups of data can clearly be seen.

We employed similar techniques to embed the features of the structure/form components of the ingredients. This feature space was reduced from 100 dimensions to 5. After all these features have been calculated we concatenate them together to give us the feature vector of the nodes.

Edges

Thus far we have yet to discuss a key component of the graph representation of the recipes - the edges. Under our construction, the edges in our graph represent a sequence of actions. This decision to model the edges as sequences of actions is not an immediately obvious one. Under our initial construction, actions were represented as nodes, however this was not internally consistent - as it forced the nodes to model two fundamentally different objects (actions and ingredients). And correspondingly this structure made it difficult to have a harmonious interpretation of what information is stored in the graph's edges. Seeking to fix this we devised another data structure where each edge contained information about one action. This too had issues as it caused much of the graph to consist of empty sigma nodes *(See Figure 5*). To overcome this issue, we define one edge as a *sequence of actions*. The hope in doing such not only reduces the total number of edges and nodes in the graph, but packs more information into each edge.

Figure 5



As described above, our edges contain information on the sequence of actions applied to an ingredient or combination of ingredients (sigma node). The challenge is how to represent these sequences of actions in vector form. Our goal is to find an embedding which clusters similar action sequences together. There are two main challenges that we encountered in attempting this. First is how to group similar actions together. If we treat the actions simply as categorical data we lose a significant amount of information as each action is equally distant from one another - when that is hardly the case. For example, *dice* is much closer to *chop* than it is to *strain*. Another challenge is that edges are sequences and hence the order is important - i.e. *blend,bake* ≠ *bake,blend*.

SGT

In order to address both these challenges, we once again draw inspiration from the field of genetics. But it is important that we first introduce the basics of this algorithm so that the reader will be able to more easily follow the logic we use to transform the edge data. Sequence Graph Transform, or SGT is an algorithm which allows comparison between variable length string data. SGT captures both local and long-term dependencies - which can be adjusted by hyperparameter κ. This algorithm computes the coincident similarity or *distance* between all two character pairings for a set of strings. In this way we can transform the sequences to a metric space more applicable for doing machine learning. SGT has found a variety of use cases - such as comparing genetic sequences or identifying user behavior on the internet (recording a sequence of key/mouse strokes).

Now that we have laid the groundwork we will introduce our approach to convert these sequences of actions into a vectorized form. We first collected and categorized all the actions. From this we ascertained that we could represent any action as being some combination of any of 13 *elemental base actions* to represent a single action- much in the same way a molecule can be viewed as combinations of their constituent atoms. These 13 base actions are: *spin,cut ,press, pull, heat, cool, bring together, air, remove air, water, remove water,split* and *reverse split*. Each of these elemental actions is represented by a unique character which are the constituent blocks of any action. These characters can be combined together in a variety of sequences to create a vast assortment of potential actions. We also introduce an additional blank space character which allows us to pad the string representations - the purpose of which is to allow us to represent the intensity elemental actions. We can also similarly represent the duration of an action with the length of the string. Note that the description above is in reference to a singular action. To create a string representation of a sequence of actions we simply concatenate the string-form of each action.

<u>dice,mix</u>

CCCCD---CCCCD---CCCCD---CCCCD---CCCCD---CCCCD---CCCCD---CCCCD---CCCCD---CCCCD---CCCCD---C
CCCD---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---

<u>chop,cut,mix</u>

C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C---C—CC--CC--CC--CC--CC--
CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC--CC-----B---B---B---B---B---B---B
---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---B---

Elemental Actions - B:*Spin* C: Cut D:Press

Here B,C,D represent elemental actions and '-' is a space holder which effectively increases the distance elemental actions are from one another (this allows us to imbue the concept of intensity of actions into the embedding). Each string is composed of repeating blocks of 4 characters which are repeated twelve times (chosen for divisibility) - the purpose of which is to allow for the representation of multiple actions concurrently in a string (e.g. blend = spin + cut) as well as to lessen the effect that strings from the tail-end have to the distance overall contribution - this is analogous to having a larger dataset so the distribution is more normal.

Once we have our string representations for each sequence of actions occurring in the dataset we can perform SGT and transform our strings into a vectorized form - thereafter we apply UMAP once again to reduce the dimensionality of the data and in turn make it more tractable to machine learning algorithms (Figure 6).

Graphs

With these definitions in hand we are ready to begin our delve into the graphs. As mentioned earlier in the paper all graphs are composed of two node types - base ingredient nodes and aggregation nodes but the majority of recipes contain *functional ingredients* which are themselves ingredients. We have devised an algorithm which breaks the *functional ingredients* into their recipe form and then reinserts this graph into the preexisting structure. It is best to view this construction from the perspective of edges *(See Figure 6)*.

G = F_1 + {(e['start],e['end])|e['start]!=F_1; e['end']!=F_1 for e in G} +
{(e['start'],F_1_terminal)|e['end']=F_1_terminal for e in G} +
{(F_1_terminal,e['end'])|e['start']=F_1_terminal for e in G}

Terms: **f** - functional ingredient (a recipe often has many)

$f_V^{(i)}$ - the graph of the ith functional ingredient (from list of all functional ingredients)

G - graph

Algorithm to deconstruct graph G:

While f^k ∈ **V:** (While there exists a functional ingredient in the deconstruction)

F = f_V^k

U = {(e['start'],e['end'])| e['start']!=F_1; e['end']!=F_1 for e in G}

edges in graph G which are not connected to node f^k

S = {(F_1_terminal,e['end'])|e['start']=F_1_terminal for e in G}

nodes in graph G which start from node f^k

E = {(F_1_terminal,e['end'])|e['start']=F_1_terminal for e in G}

nodes in graph G which terminate at node f^k

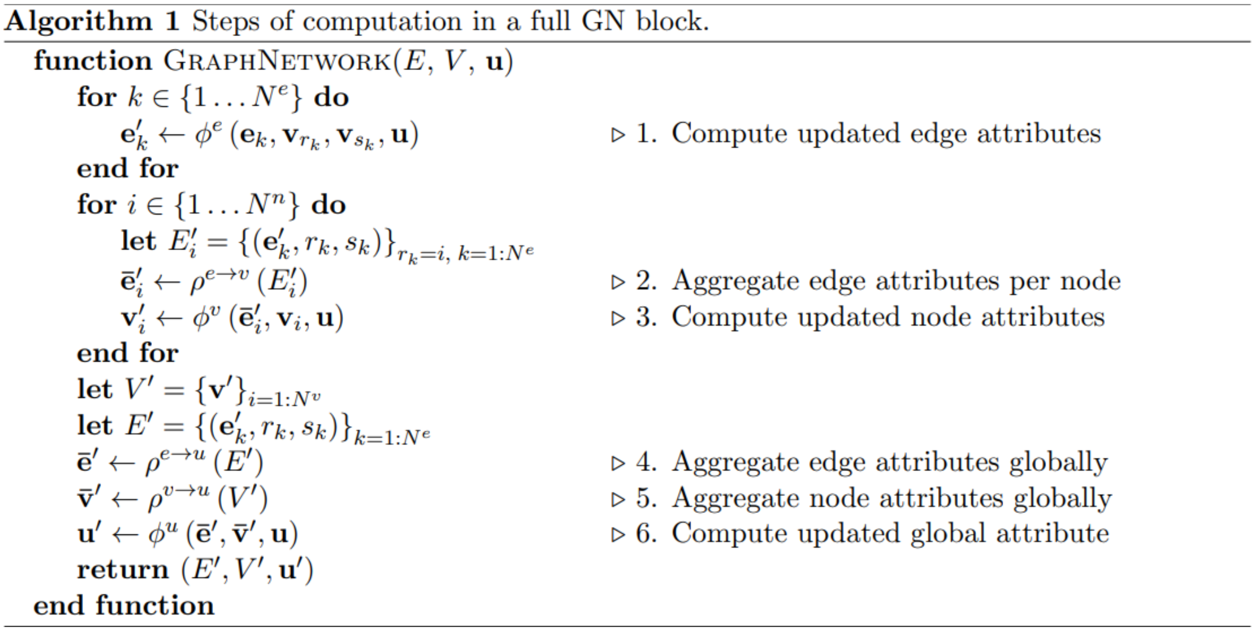
NOTE THIS E is a variable NOT the ℇ of G(V,ℇ)

G = F + U + S + E

V = G(V)

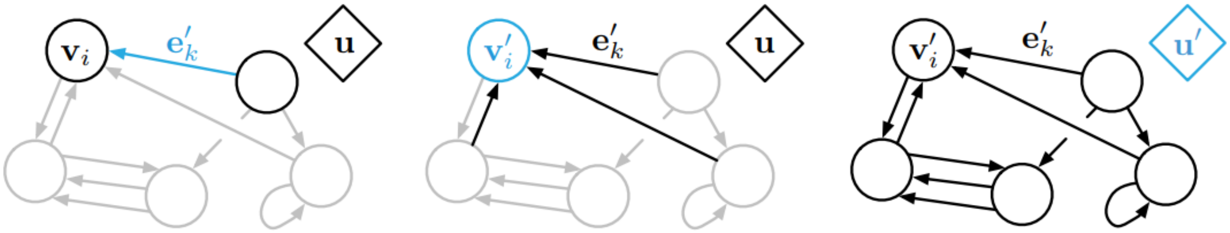
Graph Models

As a proof of concept for this graph structure, we trained two models to be able to distinguish real and fake (synthetic recipes). The first of these models is based on the *GraphNetwork* algorithm developed by Battaglia et al. (2018): Relational inductive biases, deep learning, and graph networks. This algorithm as described below is a type of message passing algorithm which successive aggregates the node, edge and global features of a graph. This hidden state representation of the graph essentially represents each constituent part of the graph in respect to its place amongst the other constituent parts - here a constituent part can be a node, edge or the global label.



Algorithm 1 Steps of computation in a full GN block.

```
function GraphNetwork($E$, $V$, $\mathbf{u}$)
    for $k \in \{1 \dots N^e\}$ do
        $\mathbf{e}'_k \leftarrow \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right)$        ▷ 1. Compute updated edge attributes
    end for
    for $i \in \{1 \dots N^n\}$ do
        let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$
        $\bar{\mathbf{e}}'_i \leftarrow \rho^{e \to v}\left(E'_i\right)$        ▷ 2. Aggregate edge attributes per node
        $\mathbf{v}'_i \leftarrow \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right)$        ▷ 3. Compute updated node attributes
    end for
    let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
    let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
    $\bar{\mathbf{e}}' \leftarrow \rho^{e \to u}\left(E'\right)$        ▷ 4. Aggregate edge attributes globally
    $\bar{\mathbf{v}}' \leftarrow \rho^{v \to u}\left(V'\right)$        ▷ 5. Aggregate node attributes globally
    $\mathbf{u}' \leftarrow \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right)$        ▷ 6. Compute updated global attribute
    return $(E', V', \mathbf{u}')$
end function
```



(a) Edge update (b) Node update (c) Global update

The second model we experimented with was a\ k-GNNs. The basic idea behind the k-GNN is for some graph *G(V,E),* for every subset of equal degree k ≤|V|, learn an embedding for each of the individual nodes - subsets of degree one -and then use this embedding to learn embeddings for each pair of nodes - subsets of degree two.We iteratively, continue this process until we reach the graph level itself and we can continue no further. In this way we build a hierarchical understanding of the nodes in the network - note this model does not use edge features in the calculation. For a more complete description of the model please refer to paper ***Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks*** Morris et al.

Overview of Update Algorithm for k-GNNs

$[V(G)]^K = \{t \in V(G) \mid |t| = k\}$

N (s) $= \left\{t \in [V(G)]^K \mid |s \cap t| = k - 1\right\}$

N_l(s) = $\{t \in N(s) | (v, w) \in E(G); v \in s \backslash t; w \in t \backslash s\}$

N_G(s) = $N(s) \backslash N_L$(s)

$$f_k^{(t)}(s) = \sigma\Big(f_k^{(t-1)}(s) \cdot W_1^{(t)} + \sum_{u \in N_L(s) \cup N_G(s)} f_k^{(t-1)}(u) \cdot W_2^{(t)}\Big)$$

Training

Following training, we find the classification capability of the model to be decent but still has much room for improvement. Further model architectures should be explored. There are a variety of issues which we believe may limit the predictive power of this model. Namely,

1. Aggregation Function for Message Passing
    a. Currently using avg(incoming nodes) - other aggregations e.g. max, RNN or some other permutation invariant function
2. Increase Feature Space Dimension
    a. Edge or Node - either add new features or increase the embedded dimensions

3. Add pertinent graph label data
    a. This is will passed throughout the network - the hope is that this feature information may help to better identify the graphs
4. Use less realistic synthetic data
    a. Synthetic data is being created by drawing from the distribution of used ingredients. The end result is that many of the “Synthetic” recipes are quite realistic. E.g. (names are made up)
        i. **STICKY CHILI** - chili powder~75%,salt~12.5%,water~12.5%,mix
        ii. **SALTY BRINE** - salt~2%,water~98%, mix
        iii. **ONION SALT** - salt~75%,onion powder~25%

Results

Using a basic form of the Message Passing model we were able to achieve 95% train accuracy and 83% test accuracy in classification. After training we find our kGNN model to have 98.4% train accuracy and 85.82% test accuracy which seems to indicate that the model is overfitting the data. We have tried reducing the epochs run and the number of convolutional layers, but these however slightly reduce the accuracy on the test data. We have also tried merging together sigma nodes which only connect to other sigma nodes - this actually decreased the accuracy. We believe this might be due to the merging of nodes causing the edges to contain longer action sequences which in turn makes them less common and hence harder to learn.

Contributions

The primary contribution of this research is the novel graphical representation of food recipes. Previous work has used NLP based techniques to attempt to create recipes however there are significant barriers - due to the computer only understanding the superficial relation between words or tokens corresponding to the ingredients some information is inherently lost. This paper hopes to address this issue more directly by organizing recipes into a standard structure where the basal level elements are natural* food items *rather* than words or tokens representing those items.

Challenges

There were multiple challenges faced in the process of converting the recipes over to a standardized format. The first challenge was determining how to represent each recipe in a generic structure - that is to identify and prescribe the most basic building blocks of the data structures representing each recipe. The final graph structure employed in this thesis represents only the most recent iteration in data structure.

As described in the DataSource section, there were difficulties parsing the original dataset obtained for this project. In order to ensure the model was trained on accurate data, we ended up manually parsing each recipe. Examples of these difficulties included non-standard writing styles, foreign words and unspecific/unclear vocabulary. Most Natural Language Models are trained on datasets such as *Reddit,Wikipedia*, etc. which is a very different style than that of a food recipe- additionally numerous recipes had additional verbiage which threw the models off- e.g. various asides about how much their *Nona* loved to cook this dish with lard instead of butter, etc. Other issues encountered included the use of foreign language words as recipes coming from different cultures often use the words of their home tongue - e.g. *jiuqu, aloo, glaçage* — similarly some words have multiple names - e.g. *coriander/cilantro*.

Further Work

Before this model can be developed for other purposes, its own accuracy and shortcomings should be addressed. Further work on this model can be broken into two categories - data related and model related.

Given that the dataset for recipes is rather small ~700 and that many recipes were transcribed owing to them being ingredients themselves in another recipe (e.g. *chicken stock, vodka, ketchup*) we still have a rather limited exposure to the almost countless varieties of potential recipes. Aside from simply expanding the dataset, we can also augment the data by adding additional label data - i.e. *Sauce,Vietnamese*. As these labels propagate through the network they may aid in positively identifying recipes. We could also further add more features to our nodes - such as nutritional information and the presence of various compounds.

Other data structures may also be explored- such as multigraphs or hypergraphs. Hypergraphs may be promising as they model connections between sets which seems naturally suited to this task. Other models should also be explored which might be better suited to modeling some of the types of relationships present in a food recipe. The first type of model that comes to mind is RGCN - Relational Graph Convolutional Neural Network - however further work needs to be done to clearly define all the types of relations.

Should this model be developed to a sufficient level of accuracy, there are a multitude of interesting use cases - such as generating synthetic recipes. One such idea is to develop a model which creates recipes which can be rated by human users which it in turn uses to further improve and refine itself.

We considered using temporal graphs to model the recipes but there were several issues with that approach. First, each recipe has multiple components which can be composed asynchronously - almost in a Gannt like fashion. Second, many steps don't have a prescribed time - e.g. chop onions. Third is that recipe times are a rule of thumb and not a clearly defined metric - e.g. cook until browned. These factors could likely be better modeled with a temporal graph, but due to the scope of this project we did not further explore this route.

We imagine several scenarios where the data structures displayed throughout this paper may see application.

Domain	**Nodes**	**Edges**
Supply Chain	Components/Parts	Fabrication Steps
Geology	Elements/Minerals	Geologic Forces

Pathology	Organic Structures	Influences/Effects

Hypergraph Structure

Like an after dinner aperitif we present to you a hypergraph structure to model recipes. The structure is similar to graph construction but differs in two significant ways.

Most obviously is that the Hypergraph Construction allows for edges connecting more than two nodes together. This in a way allows us to treat the subcomponents of the data structure more like sets (collections of items) rather than individual pairwise components. For example, in a spice blend a hypergraph may only require one edge to represent the combination of the elements. Furthermore, this edge will connect *ALL* the base ingredients and as such information will be shared more directly between the nodes. Not only does this construction reduce the number of edges but it facilitates more efficient message passing through the network.

The second significant difference between these two constructions is that our hypergraph would contain two edge types - Action Sequences and Combinations. This breakdown was chosen as the action *mix* occurred in many action sequences and had the overall effect of artificially inflating the similarity between many action sequences. That is, the embedding of {“*cut”,”bake,boil”,”stir”*} will represent the action sequences more accurately than an embedding of {*“cut,mix”,”bake,boil,mix”,”stir,mix”*}.

If we were to classify this type of network it could be categorized as a Relational Hypergraph. Aside from the differences explained above, the other components are quite similar - nodes represent base ingredients and edges represent action sequences (or combinations). *(Figure 7)*

Bibliography

1. *Digitizing National Cuisines: Cookies Recipes as Conceptual Graphs*, Dmitri Dmitriev - Institute of Linguistic Studies, Russian Academy of Sciences St. Petersburg, Russia
2. *Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks,* Morris et al. 2018
3. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. Bronstein et al 2021.
4. *Semi-Supervised Classification with Graph Convolutional Networks*, University of Amsterdam, Thomas N. Kipf and Max Welling. 2017
5. *Sequence Graph Transform (SGT): A Feature Embedding Function for Sequence Data Mining,* Chitta Ranjan et al. 2016
6. *A Survey on Deep Learning-Based Vehicular Communication Application*,Chia Hung Lin et al. 2021
7. *Understanding graph embedding methods and their applications,* Mengjia Xu 2020.
8. *Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders,* Teng Fei Ma et al. 2018
9. *A Comprehensive Survey on Graph Neural Networks, Zongwu et al. International Academy of Electronics Engineers, 2019.*
10. Can AI Create Molecules?. Drug discovery is a long and tedious… | by MIT-IBM Watson AI Lab | Medium. Accessed on June 10th, 2022
11. Network of Thrones: Recapping Season 7 and Predicting Season 8 — Benjamin Campbell (benjaminwcampbell.com) Accessed on June 10th, 2022
12. [2012.08019] Understanding graph embedding methods and their applications (arxiv.org) Accessed June 13th, 2022.

Tables and Figures

Figure 1

[2012.08019] Understanding graph embedding methods and their applications (arxiv.org)

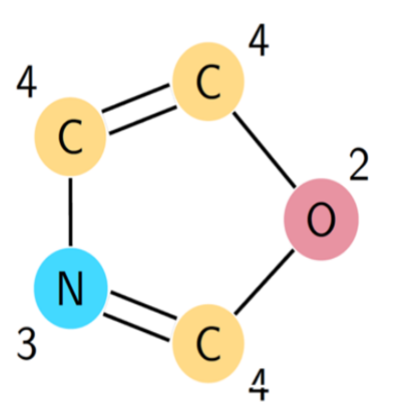
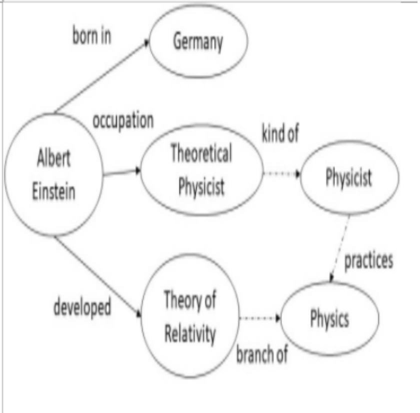
Social Graph	Molecular Graph	Knowledge Graph
		 born in Germany occupation Albert Einstein Theoretical Physicist kind of Physicist practices developed Theory of Relativity Physics branch of

Figure 2

Challenges of Graph Data

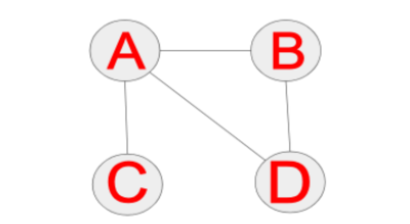
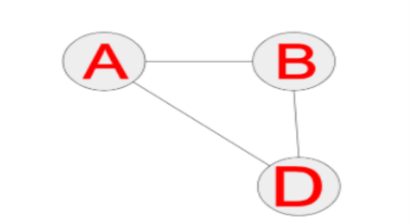
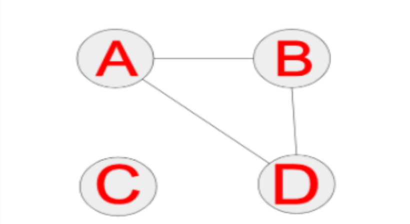
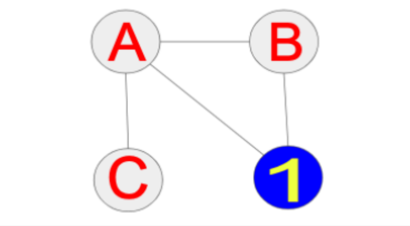
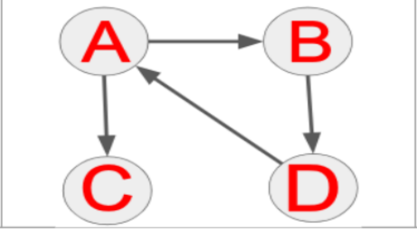
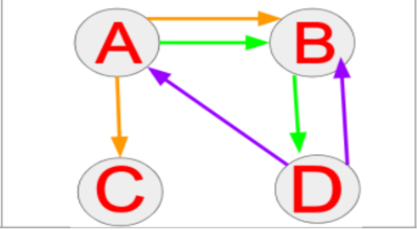
Basic Graph	Nodes	Edges
		
Node Type	Edge Direction	Edge Type and Direction
		

Figure 3

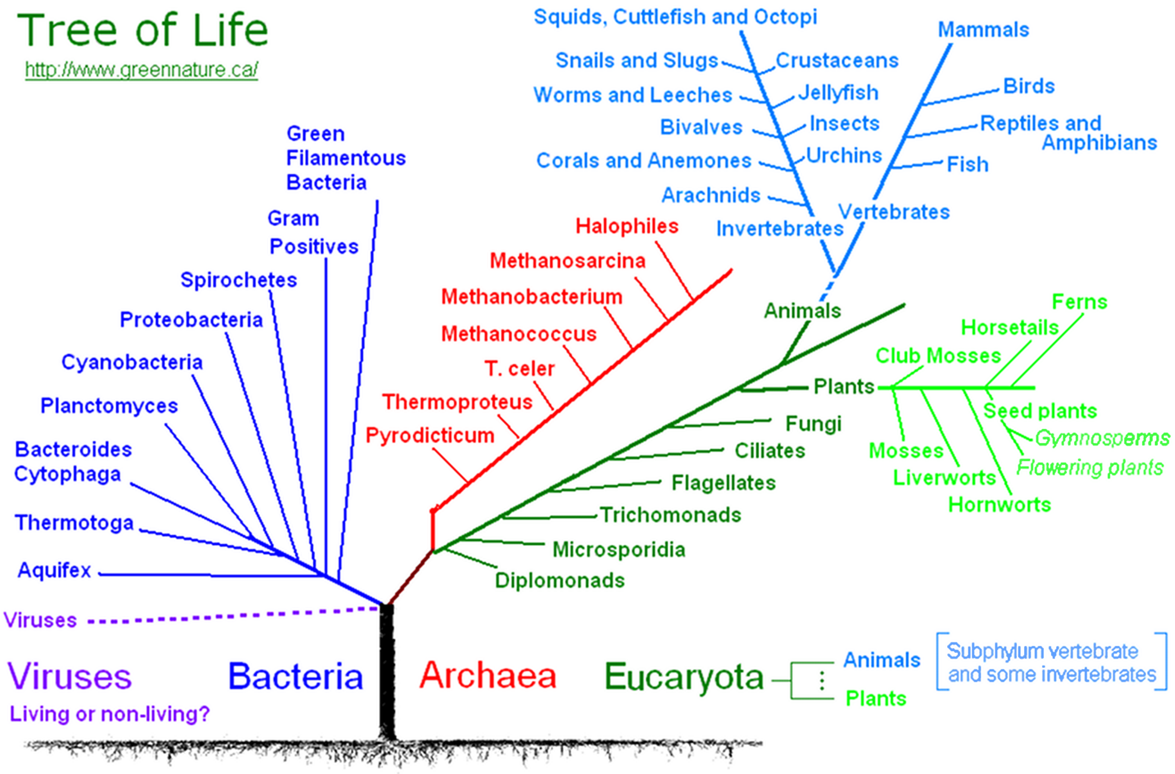
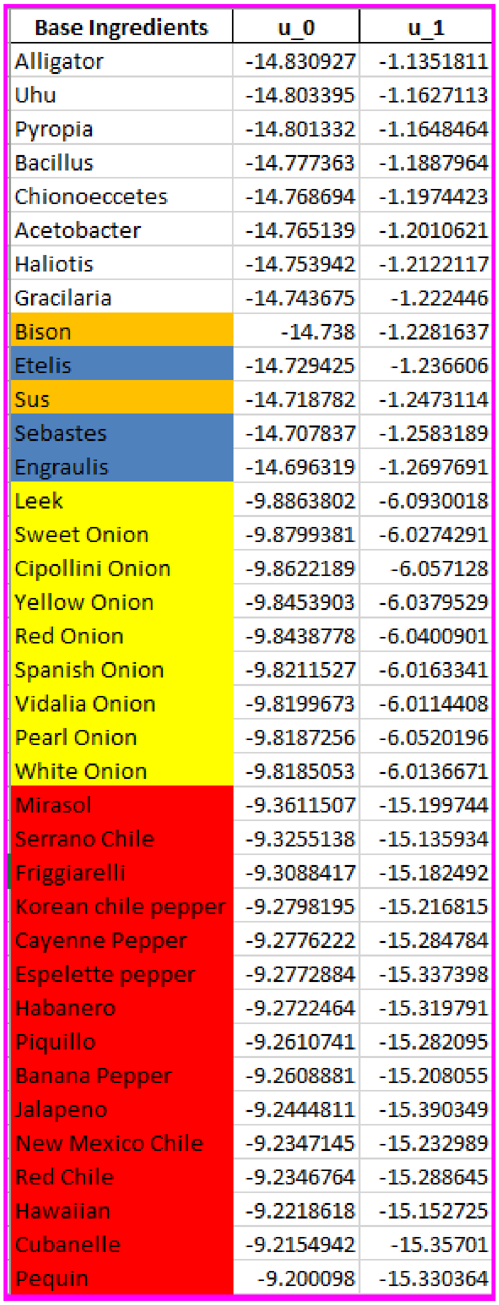

Tree of Life
http://www.greennature.ca/
Green Filamentous Bacteria
Gram Positives
Spirochetes
Proteobacteria
Cyanobacteria
Planctomyces
Bacteroides Cytophaga
Thermotoga
Aquifex
Viruses
Halophiles
Methanosarcina
Methanobacterium
Methanococcus
T. celer
Thermoproteus
Pyrodicticum
Squids, Cuttlefish and Octopi
Snails and Slugs
Worms and Leeches
Bivalves
Corals and Anemones
Arachnids
Invertebrates
Crustaceans
Jellyfish
Insects
Urchins
Vertebrates
Mammals
Birds
Reptiles and Amphibians
Fish
Animals
Plants
Fungi
Ciliates
Flagellates
Trichomonads
Microsporidia
Diplomonads
Ferns
Horsetails
Club Mosses
Seed plants
Gymnosperms
Flowering plants
Mosses
Liverworts
Hornworts
Viruses
Living or non-living?
Bacteria
Archaea
Eucaryota
Animals
Plants
Subphylum vertebrate and some invertebrates

Figure 4 - Two dimensional embedding of taxonomic tree



Base Ingredients	u_0	u_1
Alligator	-14.830927	-1.1351811
Uhu	-14.803395	-1.1627113
Pyropia	-14.801332	-1.1648464
Bacillus	-14.777363	-1.1887964
Chionoeccetes	-14.768694	-1.1974423
Acetobacter	-14.765139	-1.2010621
Haliotis	-14.753942	-1.2122117
Gracilaria	-14.743675	-1.222446
Bison	-14.738	-1.2281637
Etelis	-14.729425	-1.236606
Sus	-14.718782	-1.2473114
Sebastes	-14.707837	-1.2583189
Engraulis	-14.696319	-1.2697691
Leek	-9.8863802	-6.0930018
Sweet Onion	-9.8799381	-6.0274291
Cipollini Onion	-9.8622189	-6.057128
Yellow Onion	-9.8453903	-6.0379529
Red Onion	-9.8438778	-6.0400901
Spanish Onion	-9.8211527	-6.0163341
Vidalia Onion	-9.8199673	-6.0114408
Pearl Onion	-9.8187256	-6.0520196
White Onion	-9.8185053	-6.0136671
Mirasol	-9.3611507	-15.199744
Serrano Chile	-9.3255138	-15.135934
Friggiarelli	-9.3088417	-15.182492
Korean chile pepper	-9.2798195	-15.216815
Cayenne Pepper	-9.2776222	-15.284784
Espelette pepper	-9.2772884	-15.337398
Habanero	-9.2722464	-15.319791
Piquillo	-9.2610741	-15.282095
Banana Pepper	-9.2608881	-15.208055
Jalapeno	-9.2444811	-15.390349
New Mexico Chile	-9.2347145	-15.232989
Red Chile	-9.2346764	-15.288645
Hawaiian	-9.2218618	-15.152725
Cubanelle	-9.2154942	-15.35701
Pequin	-9.200098	-15.330364

Given limited dimensionality quite good mapping - note alligator is being mapped with seaweed - this likely because alligator is a solitary branch and it has to be grouped somewhere. As you can see it does a pretty good job at grouped like with like.

Figure 5

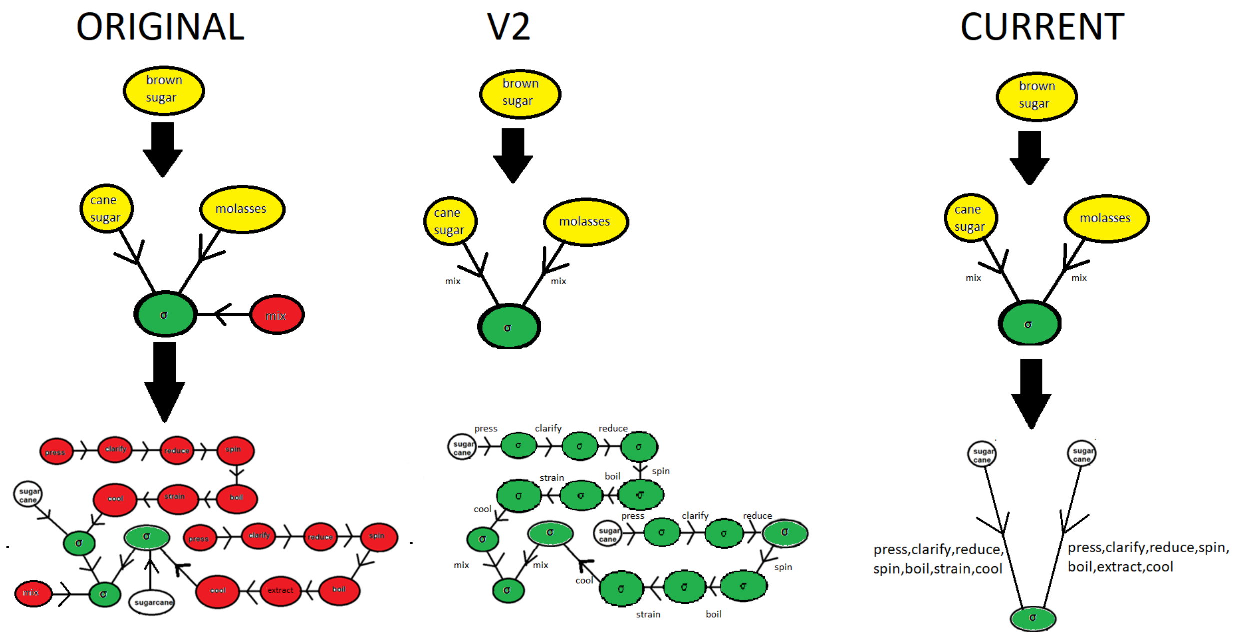


Figure 6 - Graph Representation of Soy Sauce

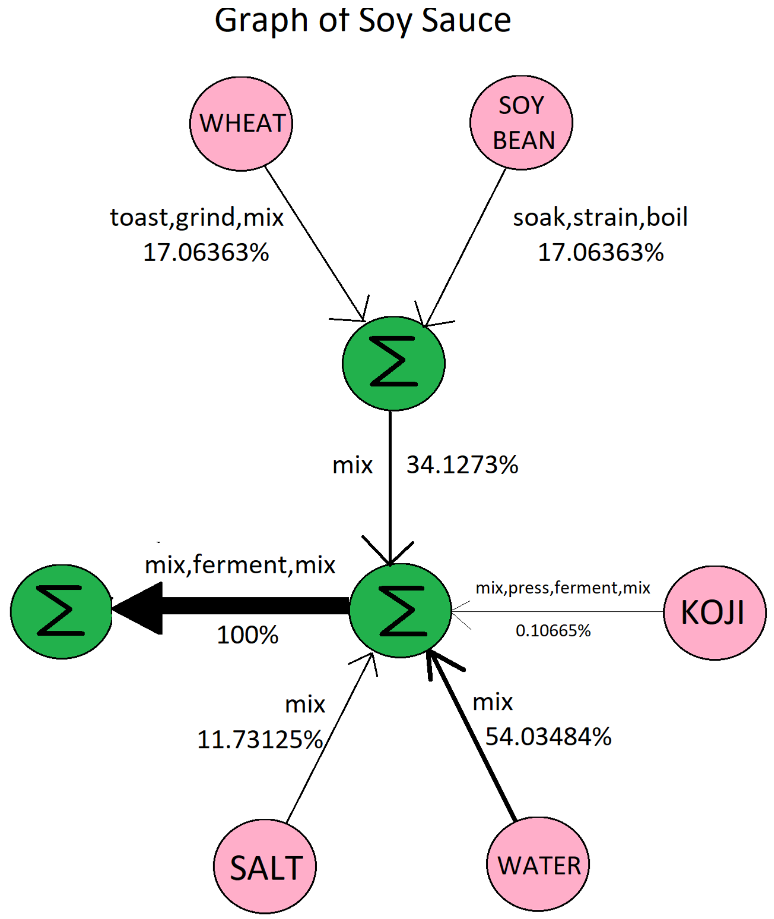
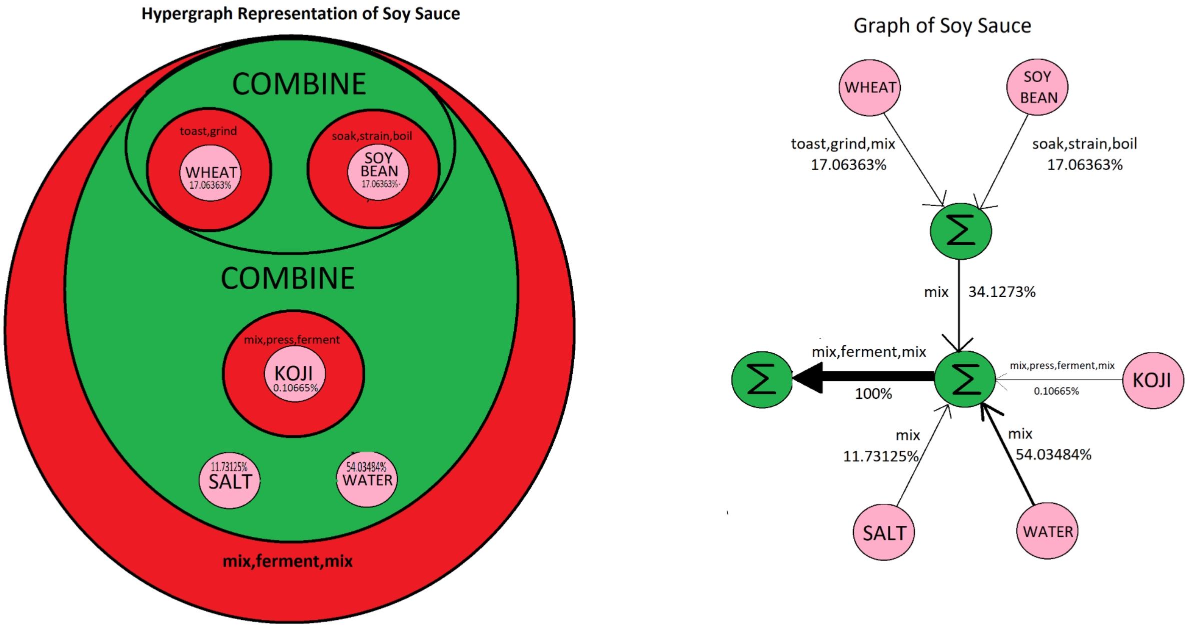
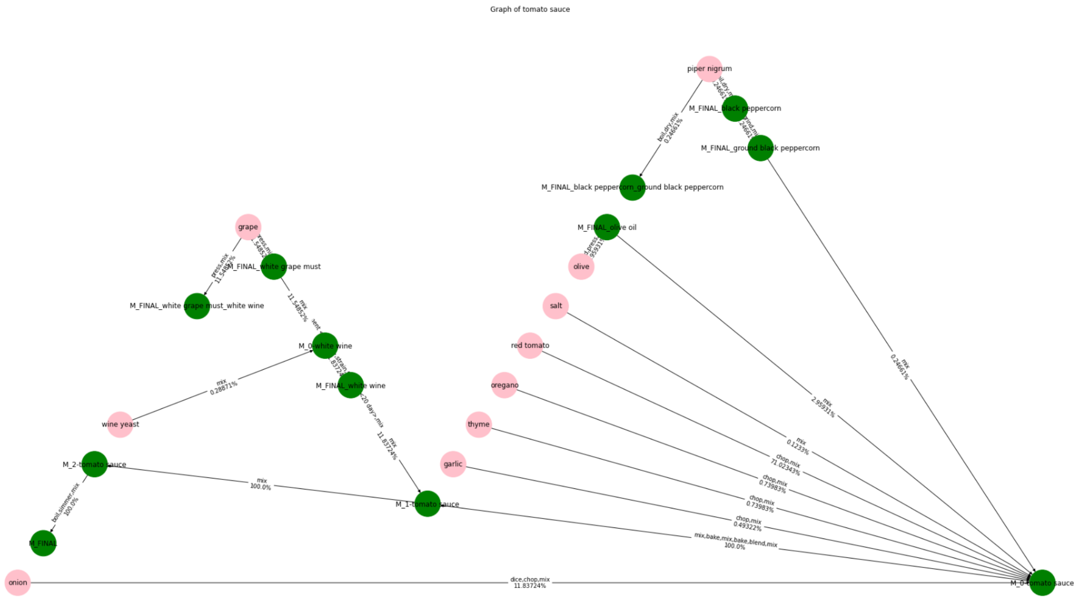


Figure 7 - Side by Side comparison of Hypergraph vs Graph Data Structure



Graph of Soy Sauce compared to potential hypergraph structure. Note to convert to hypergraph structure we will incorporate the weight into the nodes. Additionally there are now two types of hyperedges (COMBINE and action) - represented by green and red respectively.

Figure 8 - Graph Representation of Recipe Rendered via NetworkX


Graph of tomato sauce
piper nigrum
M_FINAL_black peppercorn
M_FINAL_ground black peppercorn
boil,dry,mix
0.24661%
M_FINAL_black peppercorn_ground black peppercorn
M_FINAL_olive oil
olive
salt
red tomato
oregano
thyme
garlic
grape
press,mix
11.54852%
M_FINAL_white grape must
M_FINAL_white grape must_white wine
mix
11.54852%
M_0-white wine
mix
0.28871%
M_FINAL_white wine
<20 day>,mix
mix
11.83724%
wine yeast
M_2-tomato sauce
mix
100.0%
M_1-tomato sauce
boil,simmer,mix
100.0%
M_FINAL
onion
dice,chop,mix
11.83724%
mix
0.24661%
mix
2.95931%
mix
0.1233%
chop,mix
71.02343%
chop,mix
0.73983%
chop,mix
0.73983%
chop,mix
0.49322%
mix,bake,mix,bake,blend,mix
100.0%
M_0-tomato sauce