Cooking on (Weighted) Average

Shachar Resisi*, Omri Ben-Dov†, Roy Friedman‡
Group 26

Abstract

Cooking in this millennium is a difficult chore. This is, of course, due to the overwhelming number of similar recipes that can be found by a simple Google search. Moreover, reading an entire recipe is tedious - so many lines of instructions, steps and a complicated structure that appears among pictures of how whatever you're making should look. In this project, we try to ease these day-to-day struggles. Given a recipe name, we go over multiple recipes that can be used to prepare the dish, combine them in a 'clever' way and provide a simple visualization of the new (and hopefully delicious) recipe.

1 Introduction

1.1 Problem Description

When reading a recipe for the first time, we often want to understand it quickly. Unfortunately, recipes are generally composed of long complex sentences (such as this one) describing how to combine the different ingredients and which utensils and appliances to use. Visual representations of data are usually easier to understand and less cluttered than text. The problem remains to find the most comprehensive depiction of the recipe.

An additional problem we encounter while searching for recipes is the vast number of results that match our query. Theses results are almost identical, varying in attributes such as the quantities of ingredients, the number of ingredients, and additional spices or sauces. Ideally, we wouldn't want to have to compare between all of these results, but have only one result that fits our expectations best.

These two problems are not necessarily mutually exclusive. Indeed, if a recipe of a certain dish can be represented simply, it is much more likely to find connections between that recipe and another of the same dish.

1.2 Data

For the scope of this project, we chose to focus on cake recipes. This choice can easily be changed to other types of dishes. Our data consists of thousands of cake recipes in English. These recipes were obtained by crawling the *allrecipes* website for all of their cake recipes[1] - more than 3,800 different recipes. From each recipe we extracted the required ingredients, the directions, the title, its rating, the number of people who rated it, the number of people who made it and the number of servings the recipe yields. Each recipe was stored in a single JSON file, resulting in about 15MB.

1.3 Simplifying Assumptions

In order to adjust the problems described above to a feasible course project, we simplified them a little. Our simplifying assumptions include:

1. Linearity of actions - we assume that no actions take place in parallel, i.e. we do not expect recipes to contain instructions such as "Prepare this, set it aside and prepare that". As a result, we assume that each step in the recipe refers to the previous step. This assumption is sensible in most simple cake recipes.
2. In order to understand a recipe, it is sufficient to use only directions that include at least one ingredient, cooking utensil or device.
3. All recipes are in English (and typos are ignored)
4. Each ingredient appears in exactly one direction

2 Methods

2.1 DAG Visualization of a Recipe

Usually, directed acyclic graphs (DAG) are easily readable and convey a flow in the network they represent. As a recipe is basically an algorithm, we thought that a natural representation of it is by converting it to a DAG, where each ingredient is represented by a node (e.g. one node for

*shachar.resisi@mail.huji.ac.il, shacharr
†omri.ben-dov@mail.huji.ac.il, omribd
‡roy.friedman@mail.huji.ac.il, roy.friedmam
[1] `https://www.allrecipes.com/recipes/276/desserts/cakes/`

'white sugar', another for 'all-purpose flour' etc.) and each direction is an edge that connects between an ingredient and the step in which it is used (which is also represented by a node). Due to our simplifying assumptions, each action is an edge starting in an ingredient or the result of the previous direction, and ending in a new node (which will be the start of the next direction).

Overall, our algorithm is composed of the following steps:

1. Identifying and reading the requested recipe (saved as a json file)
2. Cleanup:
   - Convert ingredients quantities to decimal form
   - Remove parenthesis in the ingredients section (as they usually don't provide additional information)
   - Separate complex directions to atomic sentences
3. Identify preparatory actions on ingredients, which are not part of the directions (e.g. "50 grams of melted chocolate" is identified as a direction to melt the ingredient "50 grams chocolate"); referred to as 'pre-actions'
4. Find the verb that best describes the action that corresponds to each ingredient
5. Identify the cooking appliances that are used, and how they are used (e.g. at what temperature the cake is baked and for how long)
6. Build a DAG

We shall now explain the non-trivial steps in detail. An illustration of this process can be found in Figure 1, and an example of the outcome of this algorithm can be found in Figure 3.

Creating the Pre-actions

Pre-actions represent the state in which the ingredients are used in the recipe, and as such do not appear explicitly in a direction, but are assumed to take place before the recipe begins. Mostly, these actions are verbs in past tense ('melted butter'), and multiple such actions can appear for each ingredient ('baked and cooled pie crust'; 'chopped bananas, mashed'). To identify these verbs, we used the NLTK PoS-tagger[2]. Each of these verbs was converted into an edge from the ingredient to itself in the final graph visualization. This course of action was found to work best, though it is limited by the tagger's performance.

Connecting Ingredients and Directions

To find in which step an ingredient is used, we first have to find its "actual" name, i.e. the name that is used to describe the ingredient in the rest of the recipe. This is due to the fact that for many ingredients, only a fraction of their full name (the one supplied in the ingredient list) reappears in the directions (e.g. 'all-purpose flour' is regarded simply as 'flour' in the rest of the recipe).

We do this by scoring the match between the ingredient and each instruction using a variation of cross-correlation. Cross-correlation is done by "moving" the ingredient's whole string along each instruction string and summing the number of matches in each step. Our variation is that for each step in this "movement" we find the length of the longest chain of consecutive matches instead of the total number of matches. A simple example of cross-correlation and longest chain calculation can be found in figure **??**.

The score for each instruction is the maximal chain length over all steps of the cross-correlation. This instruction is the instruction that uses the ingredient, and the ingredient's "actual" name is the respective longest chain.

This results in finding the longest common n-gram without a prior definition of n. Understandably, our method can be slightly modified to be able to neglect small typos. We found it redundant for our purposes (not many instances of typos in the recipes we used were found).

We furthered improved this method using TF-IDF. We calculated the TF-IDF score of all words in the recipes, considering a whole recipe (both ingredients list and instructions) as a document and stored the scores. With this table, we multiplied the match score of each instruction (length of longest matching chain) by the TF-IDF score of the matching sub-string. We found it to give better results, as matches of long frequent words would get a lower score (for example, giving priority to 'flour' over 'all-purpose').

After finding each ingredient in the instructions, we replace it with a token to decrease the number of tagging mistakes that might arise from the ingredients name, e.g. the 'baking' from 'baking powder' might be considered an action taken on an ingredient appearing later in the same step. Finally, we used SpaCy's PoS-tagger to identify the verb closest to each ingredient. Though found to give the best predictions, this method is not error-free. For example, SpaCy's tagger sometimes considers "ingredient10" (the replacing token) to be a verb. Moreover, it might have been better to calculate TF-IDF not over all of the recipes, but rather to disregard those that have weird stuff, such as bible references to scriptures where the ingredient is mentioned[3].

Identifying Appliances

A visualization of a cake recipe must include the baking of the cake. Similarly, there are other processes that might be integral to the completeness of the recipe which don't include incorporating ingredients but rather the use of kitchen appliances - cooling, baking, refrigerating, freezing, frying etc.. To properly handle these processes, we came up with

[2] https://www.nltk.org/book/ch05.html
[3] https://www.allrecipes.com/recipe/7678/scripture-cake/

a list of the most common appliances, and looked for them along with the other ingredients. If they are found, then they are added as an additional ingredient (and treated the same, i.e. connected to the corresponding actions).

As we are fans of the metric system, when baking instructions in Imperial units (Fahrenheit) were encountered we transformed them into Celsius and treated this as the needed action ('bake at 180 C' instead of 'bake at 350 F').

Creating DAGs

In order to generate graphs, we used the Graphviz package for python. We decided to generate two separate graphs - one that only contains the ingredients and actions, and a second graph in which the relevant steps of the recipe are translated into sub-graphs, that also incorporate the atomic instruction containing the ingredients appearing in this step. Example of both of these graphs can be found in Figure 3.

In these graphs, as explained in the beginning of this section, nodes stand for ingredients (or appliances), edges describe what actions should be applied to the same ingredient (where loops are a shorthand for the pre-actions that have to take place). The numbered nodes represent in which order the ingredients should be incorporated into the dish. Ingredients that are added at the same time have edges going into the same numbered node (the fact that this order exists is a consequence of our assumption that the recipes are linear).

2.2 One Recipe to Rule Them All[4]

Next, we wanted to create an algorithm for combining multiple recipes into a single, coherent recipe. The main problems facing us are that different recipes use different ingredients, a varying number of ingredients, attribute different actions to each of them, and instruct to add them in different points of time (step numbers). To handle these challenges, we did the following:

1. Identification and parsing of all of the relevant recipes (parsing includes performing steps 2-5 of the algorithm for a single recipe, as described in section 2.1); we'll denote this group of recipes as $\mathcal{R}$
2. Defining a score for each $recipe \in \mathcal{R}$, as

$$score = \log(1 + r \cdot n_b \cdot n_r) \tag{1}$$

   for r the rating the recipe received, n_b number of people that reported baking it and n_r number of people that rated the recipe.
3. The structure of the new recipe:
   - The number of incorporated ingredients is the weighted average of the number of ingredients for all $recipe \in \mathcal{R}$; we denote this number by k
   - For each ingredient we defined it's score as the sum of the scores of the recipes that it appears in. The ingredients used in the new recipe are the k highest ranked ingredients
   - The quantity of each of the k ingredients is the weighted average of its quantities in $\mathcal{R}$ (quantities were normalized according to the number of servings and the units of measurement)
   - The action associated with each of the k ingredients is the most common action performed on the ingredient in $\mathcal{R}$. Additionally, the preaction for the same ingredient is defined as the most common preaction for the ingredient in $\mathcal{R}$, taking into account the empty preaction as well (i.e. there is no preaction associated with the ingredient)
   - The step number in which each of the ingredients is added is decided according to the weighted average step number
   - The cooking time for the integrated recipe is, once again, the weighted average cooking time of $\mathcal{R}$
4. Create a DAG of the new recipe

An example of the combination of all 35 distinct 'Banana Cake' recipes can be found in Figure 4. Figure 4(b) shows a word cloud of the ingredients in all 35 recipes, and Figure 4(c) shows pairs of ingredient and its attributed verb.

Other Approaches

There are a many different approaches to combining many recipes into one. The approach as detailed above works well under the assumptions we made about the recipes in the first place. Other possible approaches, outside the scope of this project are:

- Recipes of similar dishes should have similar tastes. An approach that could work is to create a euclidean "taste" representation of each ingredient in the recipe and define the dish by the sum of these representations times the ingredients quantity, creating a euclidean representation for the dish. We tried to do this by using a Word2Vec model that was trained on articles from Wikipedia, hoping to create a vector of the ingredient's similarity to a small number of keywords. However, we found that the training of the model is utterly irrelevant for baking cakes, e.g. 'chocolate' is closer to 'bread' than 'sweet' and the similarity between 'chocolate' and 'sweet' or 'chocolate' and 'salt' is not that different
- Another approach that could work is to find sets of items that appear together and create steps from these sets of items, under the assumption that (for cakes) all of the dry ingredients are mixed together and all of the wet ingredients are mixed separately. However, in this approach giving different weights to different sets and choosing which sets (and how many) to use is not straight forward

[4] "... one script to find them, one function to bring them all and in the darkness bind them"

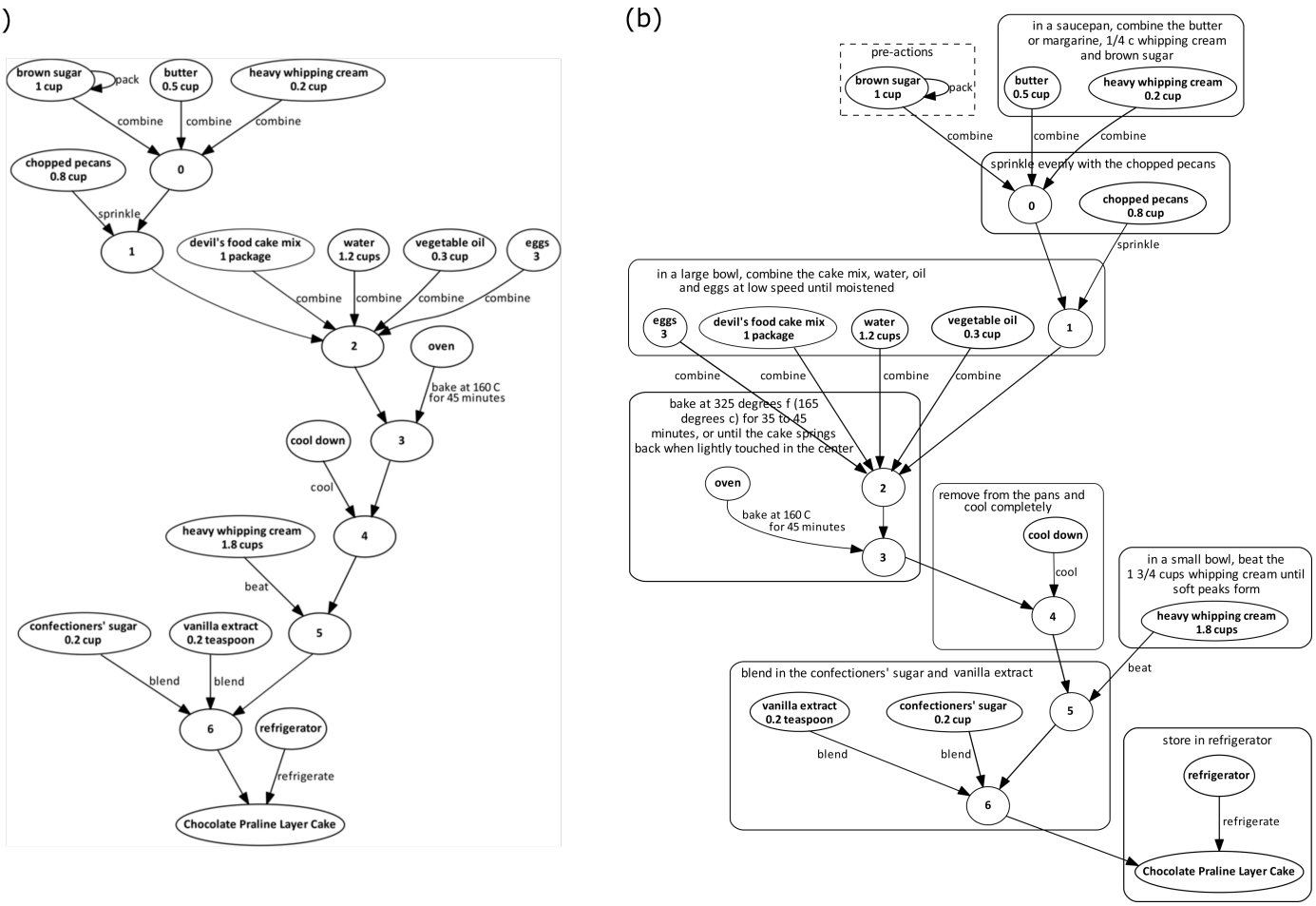

Figure 3: **DAG representation of a Chocolate Praline Layer Cake recipe**
(a) Simple graph representation. (b) Detailed graph representation, including the full directions and sub-graphs for each step.

2.3 Model Limitations and Edge Cases

The structure we assumed according to our simplifying assumptions (see section 1.3), in which we connect between ingredients and their single corresponding direction assumes that a portion of the ingredient's name appears in the direction. Interestingly, we found that this assumption fails in two general cases. The first includes recipes in which ingredients are regarded differently throughout the recipe, e.g. 'chocolate wafers' in the ingredients list changes to 'cookies' in the directions. Our model obviously fails on such cases by not finding an action for ingredients based of the ingredients list. The second case includes recipes that have general instructions, e.g. 'mix all of the dry ingredients' or 'add remaining ingredients', without specifying the ingredients explicitly. In both these cases, we have no way to link between the instruction and the ingredient. We chose to overcome these problems by introducing a default verb ('mix' was found to work well for cakes) that is linked to an ingredient that was not matched to any direction.

Another limitation arises from the cross-correlation. An additional mistake in finding the correct ingredient is when, for instance, the ingredient is 'cake flour' and one of the steps contains 'mix in the flour...' while another is 'remove the cake from the oven...', then instead of choosing 'flour' as the main ingredient name, 'cake' is considered the ingredient and added in the wrong step. This is due to the higher TF-IDF score that cake has, however we found that the addition of the TF-IDF does, frequently, help find the correct ingredient names.

Two other limitations arise from the implementation of the SpaCy PoS-tagger (though it still gave favorable results to NLTK's PoS-tagger). In some cases the tagger didn't find a verb in direction in which the ingredient appeared, even though a verb indeed appeared in the sentence (for instance, when the action was to 'beat the eggs', the verb 'beat' was tagged otherwise). We treated this edge case the same as was previously described, by linking the default verb 'mix' to any lone ingredients. The second edge case involves a specific instruction - 'grease and flour' (a pan). Though 'flour' is indeed a common ingredient name (for cakes, anyway), its role in this instruction is a verb and we would want the ingredient to be linked with a different instruction. However, the PoS-tagger does not perceive this appearance of 'flour' as a verb, which caused a recurring

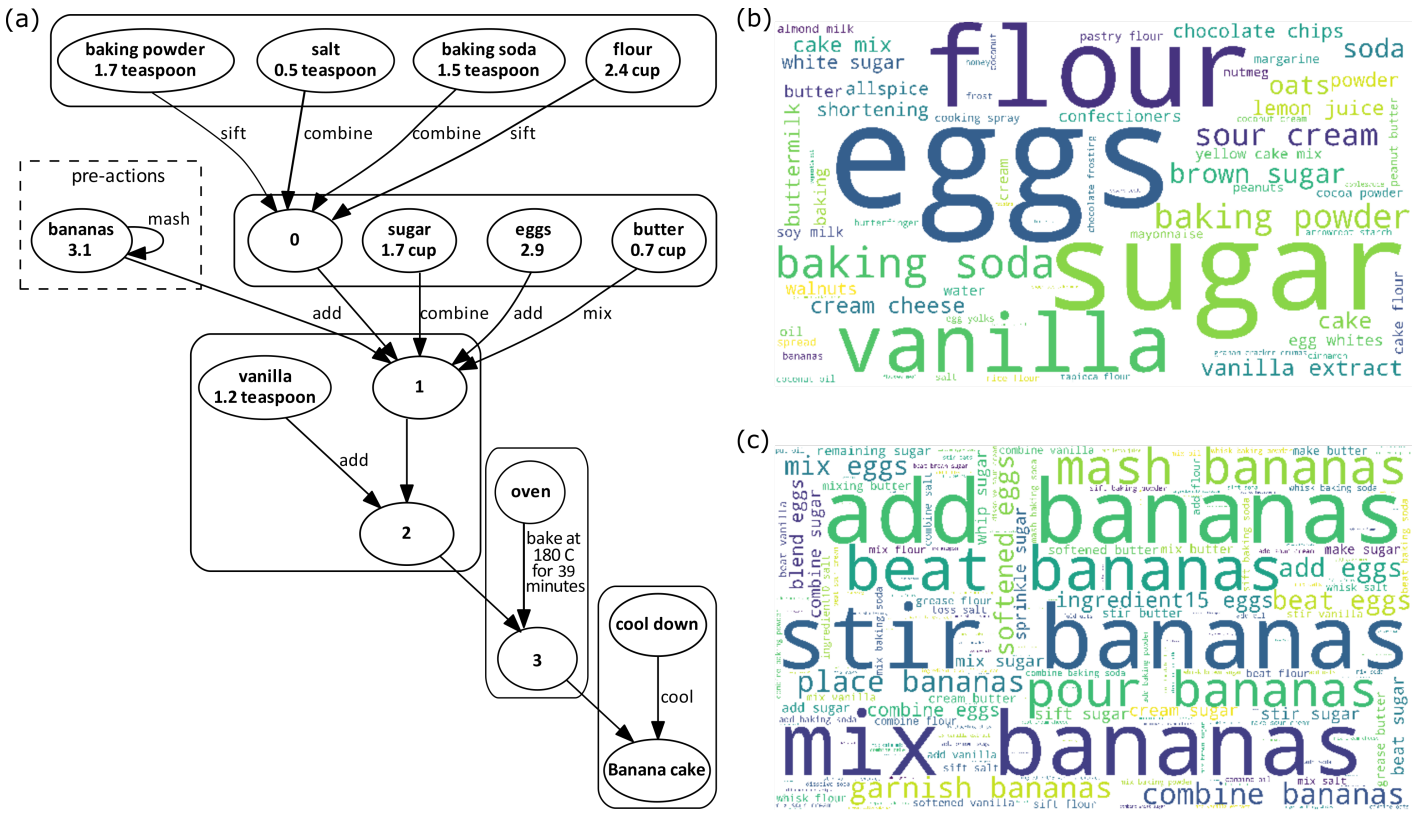

Figure 4: **Combination of 35 different 'banana cake' recipes**
(a) Graph representation of a new recipe, created by combination of 35 'Banana Cake' recipes. (b) Word cloud visualization of the ingredients in all original recipes. (c) Word cloud visualization of ingredient-attributed verb pairs of the original recipes.

problem in our representation of recipes. To overcome this problem, whenever a string of 'grease and flour' is encountered, it is simply replaced with 'grease'.

3 Results

3.1 Evaluation

The most interesting metrics by which we thought that we should evaluate our methods is their comprehensiveness, which translates to (a) how easy they are to read and follow, (b) do they convey all the essential information for making the recipe, (c) does this representation replace the need for a full recipe and (d) can a cake be made by following them. To evaluate the comprehensiveness of our DAG visualizations of recipes, we performed several tests:

1. Manual comparison of DAGs with original *allrecipes* recipes (to assess how similar they are)
2. Creation of DAGs from general recipes that were already used to bake cakes, and saw if a trail in the graph gives all the relevant information [5]
3. Preparation of the 'Chocolate Chip Cookie Dough+Cupcake=The BEST Cupcake. Ever.' recipe according to the detailed DAG (without using the textual representation of the recipe)
4. Evaluation of the compression ratio we achieve for recipes. Since we only use directions that contain ingredients/appliances, we use only a fraction of each recipe's directions. We define this fraction as the compression ratio, and a histogram of it can be found in Figure 5. In average, the average compression ratio in the number of steps was 0.56 with standard deviation of 0.12.

Naturally, these manual tests only covered a sample of the data. In addition to these tests, we also used a control group of people and asked for their criticism, and how likely they are to use the graph representation rather than the textual recipe, when making a recipe. Our control group was comprised of an artist, a medical doctor, a medical student, 4 CS majors, a Maths major, 2 Physics majors and a CS professor (a total of $n = 11$ people). All subjects showed excitement with this new representation. However, we found that graphs are not as easily read by medical professionals as by CS and Maths graduates.

[5] See e.g. https://notsosweetmeirav.blogspot.com/2019/02/blog-post.html

3.2 Visualizations

The prime goal of our project was to create graph visualizations of recipes. These results are shown in Figures 3 and 4(a). In addition to these graphs, a nice visualization that conveys a lot of information is a word cloud. Figure 4(b+c) shows the word cloud of all of the 'Banana Cake' recipes (total of 35 distinct recipes). These visualizations show the common ingredients (Figure 4(b)) and the common way the ingredients are used in the recipes (Figure 4(c)). An advantage of adding the verb to the ingredient, as apparent in Figure 4(c), is that the main common ingredient between the recipes (in our case, banana) is easily recognizable.

3.3 Run Our Code

Before running the program, the user should execute the "setup.sh" script. This script performs the following actions:

1. Downloads the recipes database to a local folder (from a public git server)
2. Sets a virtual environment to install the required packages
3. Installs the required packages in the virtual environment
4. Downloads files needed for the NLP packages

After executing this script, one can call the 've_recipy.sh' wrapper script, which activates the virtual environment and starts our program. Alternatively, if all of the required packages are locally installed (i.e. the virtual environment isn't needed), one can run the 'recipy.py' script with no additional parameters.

The script will then ask for a recipe name (for example, 'chocolate' or 'banana cake'). After inserting one, the user can either choose one of the existing recipes (with that name), or ask the program to combine them all. Choosing a single recipe will produce a list of the search results (i.e. recipes that contain the requested name), from which the user can choose. The chosen recipe graph is then created, saved and displayed. An example of the results of this process can be seen in Figure 3.

Choosing the combination option will further ask the user whether a word cloud should also be displayed. The combination process might take a while, and at its end the resulting graph and possibly word cloud will be presented, as can be seen in Figure 4 for a combination of 'banana cake' recipes.

4 Future Work

There are many possible ways to extend our work, some of which are:

- Use all of the "important" instructions, not only those that contain an ingredient. A straightforward implementation would include adding the main verb of each direction to the graphs, but it requires a better PoS-tagger than the one we use (or a smarter method for detecting cooking actions)
- Train a Word2Vec network on relevant data, and use it to find similar recipes or merge between them
- As requested by avid users, adding a feature that displays which cooking utensils are needed to complete the recipe as well as when to use each utensil
- Support recipes containing parallel actions
- Use dependency trees to choose which action needs to be performed on which ingredient. For instance, 'Combine cake mix and sugar' should output the (ingredient, verb) tuples: ('cake mix', 'combine'), ('sugar', 'combine'). Currently, the output is ('cake mix', 'combine'), ('sugar', 'mix')
- Combine only recipes with similar 'nature', e.g. exclude non-vegan recipes when combining vegan recipes
- Generalize this work to create visualizations of other algorithm types, which can be easily understood by graphs, such as code flow or anything that can be described as a set of "actions" applied to a set of "ingredients"

5 Conclusion

In this project we attempted to provide easy to read and comprehensible visualizations of recipes, specifically for cake recipes. We succeeded in doing so, under some simplification of this rather complex problem. During the course of this project, we have encountered and faced several challenges, as described in previous sections. Creating a simplified representation of recipes has allowed us to formalize and implement an algorithm that combines many recipes into one, hopefully significantly simplifying the cooking experience for future users.