# Ingredients to Recipes: CSP Generated Instructions

Kaylee Bement, Henry Lin, Nikki Taylor

March 25, 2019

## 1 Introduction

Food is the cornerstone of everyone's life. Though eating is necessary for survival, it is also a social activity, and cooking itself is seen as an art form. While only 25.9% of all food spending in the United States was on food consumed outside of the home in 1970, this percentage increased to 43.1% by 2012, due to the increase of two-earner households and the convenience of fast food and takeout options [8]. Additionally, with the rise of food delivery apps such as Postmates and Grubhub, many people are eating food from restaurants at home [5]. However, these options are both more expensive for households with greater than one person and less healthy than shopping for food and cooking it yourself [3][6].

In order to improve the convenience of cooking at home, our project aims to provide an optimal recipe for users given a set of ingredients. Though there are many options online to search for available recipes given a set of ingredients, a unique combination of ingredients may not be found in an existing recipe. Thus, our project uses a Constraint Satisfaction Problem (CSP) to learn from existing recipes in order to produce unique, creative recipes so anyone can create a meal using anything they have in the kitchen.

## 2 Related Work

There have been several previous applications of AI techniques to the overall field of recipes, but there have been very few attempts at instruction generation and fewer, if any, attempts at modeling recipes using a CSP.

One of the primary issues encountered when attempting to generate instructions is the syntax of words, specifically how verbs and ingredients should be paired together. In a paper published by Yejin Choi from the University of Washington, an Expectation Maximization (EM) algorithm was used to try and learn the connections between words in a given instruction. For example, their algorithm was able to learn that "oven" oftentimes comes later in a sentence with "bake" as a verb. Although they did not attempt actual instruction generation, this work shows one viable approach to help a program learn the relationships between common words in recipes [2].

While no formal paper has been published detailing his work, Andrej Karpathy has an open-source text generator based upon a recurrent neural network available on Github for anyone to try. The model itself takes in a text file as a training dataset and then attempts to output a text file that has similar sentences. When applied to recipes, it does generate a recipe with English words, but the recipe name, ingredient list, and instructions are not coherent. It is however an approach that is capable of generating both an ingredient list and a instructions list [1].

We did not find any literature detailing the use of a CSP to generate original instructions from ingredients, so we are excited to use a common AI technique for a new application.

## 3 Task Definition

### 3.1 Input Output Behavior

Given a set of ingredients, our recipe generator outputs an ordered set of instructions for a recipe. Each instruction is of the format
*<step #> <action> <ingredients> <location>*.

The set of instructions is listed in order of step number. An action is defined as a verb which normally appears in recipes, such as "cook" or "stir". The ingredients in an instruction can include 1 to $N$ ingredients, where $N$ is the number of ingredients the user has inputted. A location is defined as a common place to cook or mix ingredients, such as "bowl" or "pot", and if "bowl" is selected as the location, it is qualified by a bowl size, which could be "small", "medium", or "large".

One example of an instruction would be "1. cook pasta in pot", where 1 is the step number, cook is the action, pasta is the only element in the instruction's ingredients, and pot is the location.

### 3.2 Evaluation

To evaluate the quality of the recipes generated by our CSP, we implemented two different evaluation metrics. The first one outputs a numerical value where the higher the value, the more likely that the recipe is genuine. The second metric is a binary classifier. We train the binary classifier using a dataset that contains both real and fake recipes. We then run it on a testing dataset to see how well it performs. The higher the classifier error, the better our CSP generated recipes are. This is because a higher error means the classifier has a harder time distinguishing between real and fake recipes.

For the first evaluation function, we had each possible ordered word-pair in a recipe be a single feature, and then the weight for each feature be the number of occurrences for that pair in the training dataset. This way, the more often a word-pair appeared, the higher it would be weighted. The output for a given recipe would then be the sum of the weights divided by the number of ingredients in the recipe.

The second evaluation metric is a test of how genuine our generated recipes look compared to real recipes taken from the Kaggle dataset. For this binary classifier, we employed a hinge loss function combined with stochastic gradient descent (SGD). The features were extracted using an n-gram model and then trained using SGD. When run on the testing dataset, the ideal outcome would be a very high error rate. This would indicate that the CSP-generated recipes are fooling the classifier, and it is very difficult for the classifier to distinguish real from fake recipes. While these two metrics may be useful for automatic evaluation, recipes are inherently a product of human innovation, and thus, it is important to use human evaluation to discuss the outputs of the CSP, which we have used as well. Both metrics discussed above are restricted by the training data use, and thus cannot fully capture the innovative and sensory nature of recipes that goes beyond data.

## 4 Content and Methods

### 4.1 Dataset

For our dataset, we used the Epicurious Recipes dataset from Kaggle, which contains 20,050 recipes. The recipes are formatted as a list of JSON objects, where each JSON object is a different recipe. One of the keys in each recipe JSON is “directions”, which contains a list of steps for the recipe. In all uses of this dataset, the focus was on the "directions" portion of each recipe.

Additionally, we constructed our own lists of actions and locations for our instructions. For the actions list, we used a subset of the actions that occurred frequently in our dataset. The locations were determined by looking at the locations used in a random sampling of recipes in the dataset.

### 4.2 Oracle

Our oracle is human creation of recipe instructions given a set of ingredients. Nikki was given a set of ingredients, and then created a recipe for Henry and Kaylee. Based on the human evaluation of Henry and Kaylee, these recipes were reasonable and sounded delicious.

**Example:**

*Ingredients:* bananas, butter, sugar, flour, salt, eggs, baking soda, chocolate chips

*Oracle recipe:*

1. Mix flour, baking soda, and salt in a bowl
2. Beat butter and sugar in a bowl
3. Mash bananas and beat eggs in another bowl
4. Add bananas and eggs to butter and sugar mixture
5. Stir dry mixture into wet mixture
6. Fold in chocolate chips
7. Pour batter into loaf pan
8. Bake for 70 minutes

### 4.3 Baseline

Our baseline algorithm produces a set of ordered instructions by assigning each ingredient to a random action and assigning each action-ingredient pair to a random instruction number in the set.

**Example:**

*Ingredients:* bananas, butter, sugar, flour, salt, eggs, baking soda, chocolate chips

*Baseline Output:*

1. beat flour
2. boil butter
3. bake eggs
4. add sugar

5. pour chocolate chips
6. blend bananas
7. simmer baking soda
8. saute salt

From our human evaluations, this does not make much sense. Our first evaluation function produced a score of 6706.875, which is relatively high for 8 ingredients. This is mostly likely because all of the ingredients are commonly used together in recipes. However, given the same set of recipes, our CSP produces a recipe with a score of 13,015.

## 4.4 Constraint Satisfaction Problem

1. *Number of Instructions*

   The number of instructions for a recipe were determined by the number of ingredients presented. For instruction sets with less than 5 ingredients, the number of instructions is equal to the number of ingredients. Otherwise, the number of instructions is equal to 4.

   In the following sections, let $N$ represent the number of instructions for a produced recipe.

2. *Variables and Domains*

| Variables | Domains |
|---|---|
| Requested ingredient | (N), (N, N – 1), … (N, N – 1, …, 1) |
| Possible action | 0 to N |
| Where to add | Bowl, skillet, pot, kettle, saucepan, pan, "" |
| Where to cook | Skillet, pot, kettle, saucepan, pan, "" |
| Bowl size | Small, medium, large, "" |

   First, one variable is added for each of the inputted ingredients. The domain for each ingredient consists of tuples; one tuple only contains the value $N$, the next will contain $N$ and $N-1$ (as long as N $>$ 1), ..., and the final tuple in the domain will contain all values from $N$ to 1. The values in the tuple determine which steps of the recipe the ingredient is included in. We originally used all possible tuples containing 1 to $N$, but the resulting recipes did not follow a logical progression of ingredient use. The current design ensures that if an ingredient is used in one step, it will be used in all the following steps of the recipe.

   Then, a variable is added for each action in the common action list, with a domain of 0 to $N$. If an action is assigned to 0, it is not used in the recipe, otherwise, its assignment signifies its step number.

   The location portion of the instructions is divided into two variables, *"where to add"* and *"where to cook"*. Both of these variables' domains contain "skillet", "pot", "kettle", "saucepan", "pan", and "", while "where to add" also contains "bowl". Finally, we have a variable for bowl size which contains "small", "medium", "large", and "".

3. *Factors and Constraints*

   First, we added a binary constraint between each verb pair to ensure that no verbs are given the same instruction number. However, we did not add such a constraint between ingredients; if we had done so, the first step would always have one ingredient assigned to it, the second step would always have two ingredients assigned to it, etc.

   We also added a number of weighted factors while iterating over each recipe that contained our desired ingredients in the dataset. First, for each recipe in the dataset, when iterating over each ingredient in each step, we kept track of the previously seen ingredient. For each previous-current ingredient pair, we weighted the factor by 1.001 for each instruction step in the current ingredient that was one greater than each instruction step in the previous ingredient. Further, we weighted the factor by 0.999 for each instruction step in the current ingredient that was lesser than each instruction step in the previous ingredient. After this, we realized that there were many occurrences where every ingredient was assigned to every instruction step, so we also weighted the factor by 0.999 if the current and previous ingredient had the same tuple length. Assigning every ingredient to every step still happens often since it is logical in recipes to use all the ingredients together. However, this additional weighting ensures that this assignment is the best option, since it will be chosen despite the penalty.

   We used a similar approach for action ordering. For each previous-current action pair, we weighted the factor by 1.006 if the instruction step for the current ingredient was greater than the step for the previous ingredient. This has a higher weight than the ingredient ordering because we noticed that the order of action in a recipe had a large impact on whether or not the recipe made sense. For example, prior to this weighting, "cook <ingredient> in skillet"

frequently occurred before "add <ingredient> to skillet". Additionally, we penalized a reverse ordering of the previous and current actions by 0.999 like we did for the ingredients. We also weighted the first action we found in each relevant recipe occurring first in our recipe by 1.006 to further ensure that common starting actions such as "add" would occur near the beginning of the recipe.

When iterating over each recipe, for every action we encountered in the same sentence as one of our desired ingredients, if the ingredient contained the same step number that was assigned to the action, we would weight this action-ingredient pair by 1.002. Further, if a recipe in the dataset contained at least $N - 2$ of our desired ingredients, we weighted each action that appeared in the recipe by 1.3. This gives a high incentive to use actions that are commonly associated with the entire set of ingredients, rather than just with each ingredient on its own.

Finally, we weighted the occurrence of each location and bowl size. While iterating over each sentence that contained at least one of our desired ingredients, each time we would find an action relevant to a location or bowl size, we would look for each possible location or bowl size in the sentence. For *"where to add"* and bowl size, we would look for the occurrence of "add", "combine", "mix", "whisk", "pour", or "stir" in the sentence. If we found one of these actions, we would increase the count by one for each location and bowl size that was found in the sentence. A similar process was used for *"where to cook"*, but with the occurrence of "cook", "boil", "simmer", "chill", "refrigerate", "bake", and "toast". After iterating over all recipes, we weighted each location or bowl size by 1 + (x == location or bowl size) * number of occurrences * .001.

4. *Search*

   After the CSP was implemented, we needed a way to search for the optimal recipe. Originally, we used the backtracking search code from the Scheduling assignment. However, it took approximately 3 million operations to find an optimal recipe for 2 ingredients. We then created an implementation of beam search, which only takes between 20 and 30 operations to find an optimal recipe for any number of ingredients.

5. *Translation*

   Once an optimal assignment is found, it is originally in the form of a dictionary with the variables of the CSP as keys and their chosen value from their domains as the values. To make this more readable for humans, we created a translation function to rewrite this dictionary as recipe steps.

   *Sample output of CSP post-translation:*

   1. cook chocolate chips, baking soda, and salt in skillet.

   2. add chocolate chips, butter, baking soda, salt, and sugar to large bowl.

   3. stir chocolate chips, butter, bananas, baking soda, salt, sugar, flour, and egg in large bowl.

   4. simmer chocolate chips, butter, bananas, baking soda, salt, sugar, flour, and egg in skillet.

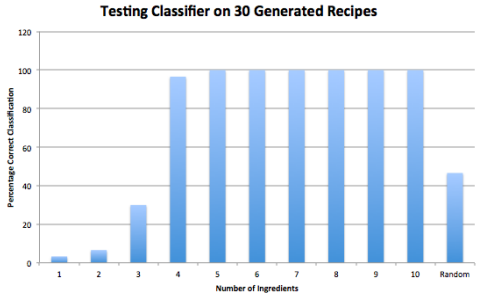
   Pre-translation, the dictionary for the above recipe would be

   ```
   { chocolate chips: (1, 2, 3, 4),
   baking soda: (1, 2, 3, 4),
   salt: (1, 2, 3, 4), cook: 1,
   cook in: skillet, add: 2,
   butter: (2, 3, 4),
   sugar: (2, 3, 4), add in: bowl,
   bowl size: large,
   stir: 3, flour: (3, 4),
   egg: (3, 4), bananas: (3, 4),
   simmer: 4 }
   ```

# 5 Experiments and Results

After testing our CSP to make sure it outputted a list of instructions properly, we wrote a script that outputted graphs using the two evaluation metrics described previously.

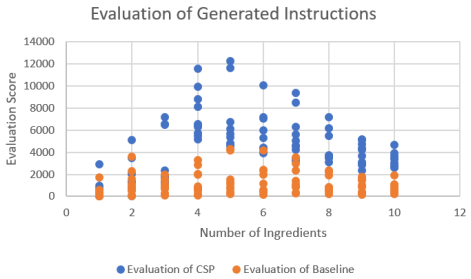
As the number of ingredients increased in the CSP model, our classifier became better at recognizing that the produced recipes were fake. Each bar in the following graph represents a different ingredient quantity, and the percentage of correct classifications for each quantity was calculated by using the classifier on 30 CSP-generated recipes with random ingredient sets. After it got to four ingredients and more, the classifier becomes nearly perfect in recognizing that it is fake recipe.




There are two likely causes for this phenomenon. The first is the varying size of the feature vector for the recipe. With a smaller ingredient list, the size of the feature vector is relatively small, meaning it is difficult for the classifier to use the majority of the information it has learned to help classify the recipe. With such a small amount of information to use, it is understandable that the classifier would perform worse.

The second reason stems from the overall coherence of a recipe. Intuitively, it becomes more and more challenging to create a recipe as the number of ingredients increase simply due to the number of variables and possibilities created with each added ingredient. We hypothesize that this is another cause of the significantly increased accuracy of the classifier. As the number of ingredients increase, it becomes increasingly difficult to model the relationship between ingredients, so the classifier is able to pick up on this trend based on the n-gram approach used. Further, most recipes do not list out every ingredient used in each step, and rather refer to groups of ingredients with words such as "mixture" or "batter", especially in recipes with a large number of ingredients. Since our CSP-generated recipes always list out every ingredient associated with a step, our classifier was able to recognize these recipes were fake as the number of ingredients increased.

However, our numerical evaluation function shows a different relationship between number of ingredients and recipe quality.




We used this evaluation function on 140 random ingredient requests and averaged the outputs to get the information shown above. The blue dots in the graph represent the output based on our CSP results, and the orange dots show the output for the baseline. On average, our CSP scores significantly higher than the baseline for recipes containing between three and eight ingredients. This shows that for recipes containing between three and eight ingredients, our CSP is able to produce instructions with common action-ingredient pairings, but for fewer or greater ingredients, it fails to find high quality action-ingredient pairs. This provides several insights to supplement our observations regarding the first graph.

First, it strengthens our hypothesis that small feature vectors cause the classifier to be unable to classify recipes with short ingredient lists, because our numerical evaluation shows that the produced recipes with short ingredient lists are actually low quality. Second, it shows that the classifier's ability to identify fake recipes with between four and nine ingredients is not due to an inability to model the relationships between ingredients, since the numerical evaluation for these recipes, which is only based on common action-ingredient pairs, is so high. Rather, it is most likely due to our other hypothesis - that our format of listing every ingredient involved, rather than using generic terms such as "mixture", makes it obvious that these recipes are fake. However, as we use more than eight ingredients, the relationship between ingredients and actions is more difficult to model, as shown in both evaluations.

# 6 Conclusion and Future Work

While the generated recipes from our CSP quickly deteriorated in quality as instruction lists increased, we are optimistic about the application of CSPs in the field of recipe generation. CSPs have primarily been used in solving

problems outside the realm of natural language process, such as resource allocation or logic puzzles, but this project shows how it can also be applied to the generation of text. We are aware of the problems associated with a CSP however, such as computational limitations, modeling difficulties, and the creation of systemic, structural text outputs.

Nevertheless, there are multiple interesting extensions possible with this recipe generator, and if given more time, it would be interesting to pursue the possibilities of using a CSP in conjunction with other methods. The way the CSP is modeled currently allows for flexibility in what constitutes an instruction, so given the computational power, it is possible to include quantity, adjectives, and other words contained in an instruction. For example, the CSP could move from generating *<step number>* *<action>* *<ingredients>* *<location>* as the instruction step format to outputting *<step number>* *<action>* ***<quantity>*** ***<adjective>*** *<ingredients>* *<location>* and thus allowing for more detailed instructions for the user to follow.

Another possible way to improve the recipe generator to better mimic human innovation is an implementation of exploration vs. exploitation in the search. The exact implementation may differ between models, but one example of this could be an epsilon-greedy approach where every so often, one of the final instructions is replaced by a random instruction. This additional feature would mimic how a cook may try something different from time to time to see what the end results are and if those end results are better than the tried and tested approach.

# References


[1] Andrej Karpathy. 2015. The Unreasonable Effectiveness of Recurrent Neural Networks. (May 2015). Retrieved November 15, 2017 from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[2] Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en Place: Unsupervised Interpretation of Instructional Recipes. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing(2015). DOI:http://dx.doi.org/10.18653/v1/d15-1114

[3] Datafiniti. 2016. Should I stay or should I go? (September 2016). Retrieved December 1, 2017 from http://datafiniti.co/eating-out-vs-eating-in/.

[4] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial Learning for Neural Dialogue Generation. (September 2017).

[5] Nutrino. 2016. Cooking vs Eating Out: The Great Health Debate. (July 2016). Retrieved December 1, 2017 from https://nutrino.co/cooking-vs-eating-great-health-debate/.

[6] Paula Martinac. Eating in vs. Eating Out. Retrieved on December 1, 2017 from http://healthyeating.sfgate.com/eating-vs-eating-out-8914.html.

[7] Sosuke Amano, Kiyoharu Aizawa, and Makoto Ogawa. 2015. Food Category Representatives: Extracting Categories from Meal Names in Food Recordings and Recipe Data. 2015 IEEE International Conference on Multimedia Big Data(2015). DOI:http://dx.doi.org/10.1109/bigmm.2015.54

[8] United States Department of Agriculture. 2017. Food-Away-from-Home. (September 2017). Retrieved December 1, 2017 from https://www.ers.usda.gov/topics/food-choices-health/food-consumption-demand/food-away-from-home.aspx.