Ingredient Recommendation System

Nikita Mohan Huikerikar 20499489
Yi Sen Wang 20500953
Bjoern Ove Sagstad 20501024
Nora Eliza Norden 20500343

Problem Definition

A situation that many home cooks might find themselves in is being at the grocery store and not knowing what to purchase. Different ingredient purchases may open up opportunities for different types of recipes. A particularly mathematically inclined home cook may wish to purchase the minimum number of ingredients while maximizing his or her potential to create different recipes. More generally, one might wish to purchase ingredients which are maximally reusable between recipes, while at the same time obtaining all the ingredients necessary to prepare today’s meal.

Design and Infrastructure

We used the dataset “What’s cooking” from Kaggle for our finished recipes, which provides us with a list of the ingredients used for any given recipe. The dataset also provides a cuisine data field, which was unused in the project. Figure 1 below shows how a recipe is read.

```json
{
  "id": 24717,
  "cuisine": "indian",
  "ingredients": [
    "tumeric",
    "vegetable stock",
    "tomatoes",
    "garam masala",
    "naan",
    "red lentils",
    "red chili peppers",
    "onions",
    "spinach",
    "sweet potatoes"
  ]
}
```

Figure 1: Sample recipe from the Kaggle what’s-cooking dataset, https://www.kaggle.com/c/whats-cooking.
The program takes a list of user defined ingredients as input, and outputs a list of ingredients to purchase, such that the user will have all the ingredients necessary for at least one complete recipe. The problem is structured as a search problem over a directed graph. The complete recipes represent end nodes in this graph. Intermediate nodes are subsets of the complete recipes, with adjacent nodes differing by one ingredient. A given node can be thought of as a set of all ingredients currently in possession for that given state. The directed nature of the edges ensures that traversing through the graph will only add ingredients. The corresponding weight for a given edge is inversely proportional to the frequency of that ingredient in the full recipe dataset. In other words, the more common an ingredient is, the lower its cost.

![Graph representation of two recipes](image)

Figure 2: A simple graph representation of two recipes.

In the above sample graph, the two complete recipes are comprised of sugar, flour, water and sugar, flour, butter. The cost of sugar and flour are both 0.5, the cost of water is 0.25 and the cost of butter is 0.75. Given an input of either sugar or flour, the program should return the recipe with sugar, flour and water (since it has a cost of 0.75, compared to 1.25 for the recipe with sugar, flour and butter).

### Implementation and Experimental Results

One issue that was encountered early on was the massive search space given the above representation of recipes in a graph. The dataset consisted of recipes using over 6700 unique ingredients. Building and running search on a graph with over $2^{6700}$ nodes would be far too computationally intensive, and thus it was necessary to constrain the search space by preprocessing the data prior to constructing the graph. This led to the design of two filters, which worked in combination to reduce the total number of unique ingredients to a more manageable size by constraining the number of recipes. The first filter kept only recipes that contain all the user input ingredients (or a subset of the user input ingredients, in the case that all the user input ingredients were not present in any recipe). The second filter was an incremental depth
filter, which trimmed recipes containing more than a certain number of ingredients more than the starting set of ingredients (the user defined ones). This number would decrease until the number of unique ingredients was below a certain threshold. This reduced set of recipes would then be used to construct the graph.

Once the dataset was trimmed down, there were two additional tasks. The first was to dynamically generate this graph from the filtered set of recipes, and the second was to run the search algorithm over the graph. Regarding graph library support in Python, we decided to use NetworkX since it includes support for both directed graphs and A* graph search. Before doing the graph search, however, it was necessary to design a heuristic.

We went with a simple heuristic that estimates the cost between a start and a goal node by returning the difference in number of ingredients.

We estimate cost by the number of occurrences and then standardize it so it’s always between 1 and 2, where the closer the cost is to 1, the more common the ingredient is, and the closer to 2, the rarer it is.

An example graph search is run with salt and cream as user inputs. The program output is shown in Figure 3 below.

![Figure 3: Example program output given inputs of salt and cream.](image)

To get a statistical foundation for the performance of our A* search, we test the program with qualitatively and quantitatively different user inputs, some of which are shown in Table 1 below. We compare the A* with UCS to measure both the optimality of our heuristic, and the improvement in computation time.
Table 1: Runtime analysis of A* and UCS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Salt, lemon juice, garlic</td>
<td>Yes</td>
<td>2.779e-4</td>
<td>4.490e-4</td>
<td>61.9</td>
</tr>
<tr>
<td>Salt, lemon juice, garlic, cumin</td>
<td>Yes</td>
<td>5.484e-3</td>
<td>6.778e-3</td>
<td>80.9</td>
</tr>
<tr>
<td>Egg, flour</td>
<td>Yes</td>
<td>6.704e-5</td>
<td>11.84e-5</td>
<td>56.6</td>
</tr>
<tr>
<td>N/A</td>
<td>Yes</td>
<td>1.191e-4</td>
<td>2.149e-4</td>
<td>55.4</td>
</tr>
<tr>
<td>Pepper, olive oil, ground turkey</td>
<td>Yes</td>
<td>3.692e-3</td>
<td>6.536e-3</td>
<td>56.5</td>
</tr>
<tr>
<td>grated carrot, black mustard seeds, curry leaves, green chilies, salt, onions, semolina, oil</td>
<td>Yes</td>
<td>9.970e-5</td>
<td>12.949e-5</td>
<td>77.0</td>
</tr>
</tbody>
</table>

Conclusion

Due to our choice of graph infrastructure, where only nodes that are subsets of other nodes and differ by one (missing) ingredient are connected by an edge, the heuristic between two neighboring nodes will always be 1, and increase linearly with the number of ingredients differing between two nodes. The cost of an edge between two nodes is standardized and in the range of 1 to 2, so the heuristic approximation of 1 is always an underestimation. This implies that the heuristic is consistent, and should thus always return the optimal path. This also holds empirically (see Table 1, column 2).
The speed improvement is quite remarkable; at best, we almost halve the computation time compared to UCS. However, for rare user inputs, or inputs yielding bigger graphs\(^1\), the A* search does not perform as well. The heuristic underestimates the costs of rare ingredients by a lot, meaning that the steps taken in the search is too small. Thus, both large graphs and rare ingredients imply more nodes to be searched. For an extreme problem, the number of nodes searched through would converge to the brute force search of the UCS algorithm.

**Further Work**

Based on our current implementation, there are a number of additional features that we envisage adding to our program.

Firstly, we could make use of the fact that our dataset provides a cuisine tag for every recipe, allowing the user to input a desired cuisine when running the search. Our program would then return results using only recipes of this cuisine. With some further data pre-processing, we could extend this idea of filtering searches. For example, manually adding tags to each recipe specifying whether they are vegetarian, vegans and so on would allow the user to run a much more refined search.

Adding a user interface will also improve the usability of the program. This would make it easy for users filter recipe inputs as well as allowing us to return additional data such as pictures and recipe instructions.

The A* search could be further optimized (in time) by using a heuristic function that better estimates the edge costs, especially for rare ingredients. This would solve the issue of taking too small steps to the direction of the goal mentioned earlier. However, execution of the program shows that building the graph takes longer time than searching it, regardless of search algorithm used. This is due to the fact that the graph is built every time. Our decision to do this was based on the fact that we could never build a complete graph with all possible nodes given all existing ingredients and recipes in our database, since we would quickly run out of memory, so the implementation is essentially a tradeoff between memory and computation time. A way to improve the program would be to look into other options than representing our problem as a graph search. Instead, we might be able to solve the problem using more modern techniques, e.g. neural networks, but that is beyond the scope of this course.

---

\(^1\) A big graph can be caused by input ingredients that are subsets of large recipes even though the number of unique ingredients and recipes are bounded.