Rheinische Friedrich-Wilhelms-Universität Bonn
Smart Data Analytics - Institut für Informatik III

Masterthesis

in Computerscience (M.Sc.)

Evolution and Generation of Cooking Recipes

Jonas Weinz, 2571421

May 15, 2020

Supervisors: Dr. Hajira Jabeen
Prof. Dr. Jens Lehmann
Dr. Diego Esteves

Rheinische
Friedrich-Wilhelms-
Universität Bonn

Prüfungsausschuss
Informatik

universität **bonn** · Institut für Informatik · 53012 Bonn

Vorsitzender des Prüfungsausschusses

Prof. Dr. Thomas Kesselheim

Prüfungsamt:
Judith König
Tel.: 0228/73-4418
Fax : 0228/73-4788
pa@informatik.uni-bonn.de
Postanschrift: 53115 Bonn
Endenicher Allee 19a

www.informatik.uni-bonn.de

Erklärung über das selbständige Verfassen einer Abschlussarbeit/ Declaration of Authorship

Titel der Arbeit/Title:

.....

.....

.....

Hiermit versichere ich, _____, _____,
Name / name, Vorname / first name

dass ich die Arbeit – bei einer Gruppenarbeit meinen entsprechend gekennzeichneten Anteil der Arbeit – selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

…… Bonn, den ……..
Unterschrift/signature date

Acknowledgements

First I want to thank Dr. Hajira Jabeen to support me during this thesis. She always had an open ear for questions and was available for discussions to find out the right direction for this thesis. I also want to thank Prof. Dr. Jens Lehmann to make it possible to work on this thesis in the Smart Data Analystics research group at the University Bonn. For support and hints in natural language processing topics I want to thank Dr. Diego Esteves. I also want to thank Carsten Draschner to share his thoughts, some hints about Evolutionary Algorithms in Artificial Culinary and some useful datasets with me. I am also thankful to everyone who participated in the evaluation surveys, without them there would be no evaluation data. Finally I want to thank all my friends and family members who supported me during the time of writing this thesis.

Abstract

A lot of cooking recipes are provided by cooking books or cooking websites. Each of those cooking recipes contains information about ingredients, ingredient mixings, cooking procedures and much more. This thesis' goal is to learn this information from existing, user created recipes and use it to create new cooking recipes automatically. To reach this goal an analysis is first performed over a large amount of user generated recipes to extract cooking related information. An evolutionary algorithm can then use the extracted information to generate valid cooking recipes on it's own.

Contents

1 Introduction and Motivation **1**
1.1 Autochef Overview and Thesis Structure 1

2 Related Work **3**
2.1 Evolutionary Algorithms 3
2.2 Evolutionary Algorithms and Machine Learning in Culinary Arts 4
2.2.1 EvoChef 4
2.2.2 SmartChef 5

3 Datasets **7**
3.1 Data Representation of Cooking Recipes 7
3.2 One Million Recipe Dataset 7
3.3 Yummly Dataset 8

4 Recipe Analysis and Information Extraction **10**
4.1 Extracting cooking related vocabulary 11
4.1.1 Classifier implementation 11
4.2 Recipe Analysis 12
4.2.1 Recipe Structure 12
4.2.2 Process Recipe States and Ingredient States 13
4.2.3 Postprocessing of adjacency matrices 14
4.3 Discussion of Data Extraction Results 15
4.3.1 Pairwise ingredient mixes 15
4.3.2 Combinations of actions with ingredients 17

5 Evolutionary Algorithm **19**
5.1 Initialization 20
5.2 Mutation 23
5.2.1 Mutation of Mix Nodes 23
5.2.2 Mutation of Action Nodes 23
5.2.3 Mutation of Ingredient Nodes 25
5.3 Crossover 25
5.4 Fitness Evaluation 26
5.4.1 Ingredient Node Scores 26
5.4.2 Action Node Scores 27

5.4.3 Mix Node Scores 27
5.4.4 Tree Score 28
5.5 Textual Tree Representation 30

6 Evaluation and Discussion of Results 32
6.1 Survey 32
6.2 Conclusion and Future Work 33

A Cooking Action Set 37

B Attached Digital Content 38
B.1 CDRom Content 38
B.1.1 Additional data 38
B.2 Autochef Python Package 38
B.2.1 Install AutoChef as Python Package 39
B.2.1.1 Example usage 40

Chapter 1

Introduction and Motivation

Cooking, or the processing of food in general, plays an important role in everyday life. So it's not surprising that there are a lot of human generated cooking recipes available in books and various cooking websites. Despite the success of information technology and it's tools (e.g. machine learning) in other fields there are not many applications available in the culinary domain so far. One example was IBM ChefWatson [1], an approach that generated new cooking recipes but is not available anymore.

Goal of this work is to extract information from user generated recipes to learn how ingredients can be processed and which ingredients can be mixed together in a cooking process. In a second step this extracted information can be used to generate new valid cooking recipes, given few initial information from a user about the type of the recipe. The idea is that a user can make a request like "I want to cook something with rice and tomatoes" depending on ingredients that are available in the user's kitchen and gets a new recipe using those ingredients as answer.

This thesis presents AutoChef, a fully automatic approach to scan user created recipes and create new ones given some initial ingredients.

1.1 Autochef Overview and Thesis Structure

In Figure 1.1 Autochef's structure is shown.

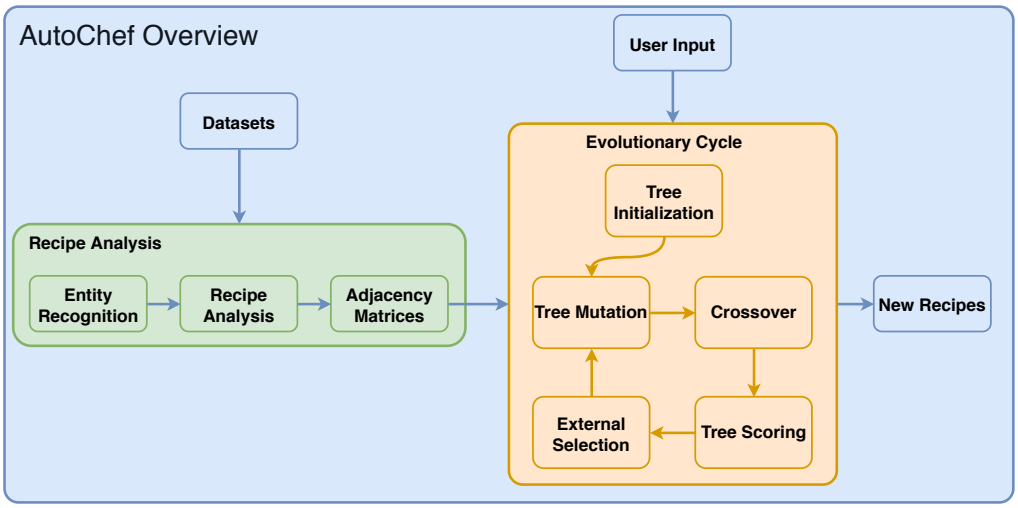


Figure 1.1: AutoChef Overview

It's mainly divided into two steps. The first one is the recipe analysis where ingredients and cooking instructions are detected and their relations extracted. The method is explained in detail in chapter 4. The result of this process is then used in an Evolutionary Algorithm which generates new recipes as output and is discussed in chapter 5. Also a reference implementation in `python` [2] is attached to this thesis' source files.

Before this thesis presents AutoChef in detail, Evolutionary Algorithms with focus on Culinary Arts are discussed in chapter 2. Futhermore the datasets that are used as input in the recipe analysis step are shortly presented in chapter 3.

To measure the approach's performance user surveys were created and evaluated, shown in detail in chapter 6, followed by a discussion of the results.

Chapter 2

Related Work

2.1 Evolutionary Algorithms

Evolutionary Algorithms try to find an optimum solution inside a search space using methods inspired by the natural evolutionary process observed in nature [3].

What an "optimum solution" is is defined by an optimization criterion, mostly an objective function $f(x) = y, f : X \to Y$ where X is the search space and $y \in Y$ gives us the optimization value that has to be maximized. In the context of Evolutionary Algorithms this optimization function is often referred as fitness function, referencing to the evolutionary fitness of species' individuals in nature.

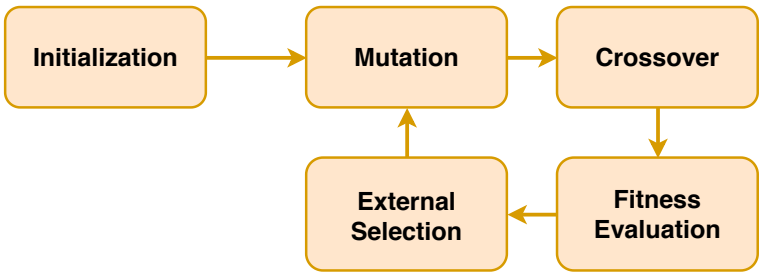


Figure 2.1: Evolutionary Cycle as it's is commonly used in Evolutionary Algorithms

As shown in Figure 2.1, Evolutionary Approaches generally consists of the populations' Initialization followed by an evolutionary cycle that can be repeated over a dynamic number of generations.

Single search spaces' instances $x \in X$ are called individuals and an individual is represented by it's genome. Often the genome is a sequence of characters or numbers, represented as vector or string. In Holland [3] for example, the genome

is represented as a binary string of a fixed length. The set of all existing individuals of a generation is called population $P \subset X$. During the mutation step each individual's genome is mutated. Either a fixed number of mutations is applied on the whole genome or a probability function defines the chance of mutation for an atomic element inside the genome. In the genome presented in figure 2.2 one way to mutate it can be a bit flip that is applied with a random chance on each bit in the binary string. It is important that the mutated individual x' remains in the search space X. Depending on the implementation either x gets replaced by x' or x' is a new individual such that both, x and x' are in the population P after the mutation step.

In the crossover step population's individuals are mated together (mostly pairwise) and then mates' genomes are combined to either create genomes for new individuals or exchange parts of their genome with each other. In the binary genome example discussed in [3] a random cut point is chosen for a genome pair and all bits after the cut are exchanged between the two individuals (see Figure 2.2 for an example).

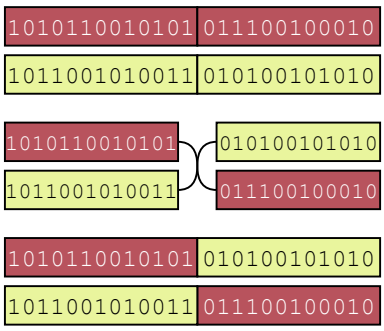


Figure 2.2: Crossover Example of two individuals as presented by Holland [3]

After the crossover the fitness $y = f(x)$ of each individual $x \in P$ is evaluated. Following the external selection takes place where the fitness values are used to determine which individuals will survive and build the next generation. There are multiple strategies for an external selection. The simplest method is to hold the n best individuals where $n = |P|$ is the size of the next generation's population. Another possible strategy is to mate individuals in pairs or groups together and let the group's winner survive. This strategy allows more exploration of the search space X.

2.2 Evolutionary Algorithms and Machine Learning in Culinary Arts

2.2.1 EvoChef

EvoChef [4] uses existing recipes, crawled from cooking websites like Yummly [5], and generates new recipes using the crawled ones. In order to process the

crawled recipes they are manually converted from a serial, textual representation to a graph (see also chapter 3.1).

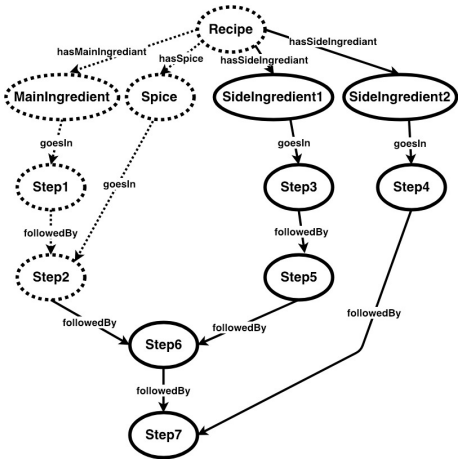


Figure 2.3: EvoChef Recipe Graph (Image from [4])

In Figure 2.3 a recipe's graph representation in EvoChef is shown. One recipe contains of a set of nodes: a main ingredient, several side ingredients, cooking steps (containing cooking instructions) and spices. The nodes are connected by edges that define the relations between the nodes. Given recipes in graph representation, EvoChef uses an Evolutionary Algorithm to genreate new recipes. Mutation is done by a random replacement of ingredients from a food substitution graph and crossover is done by a fixed point crossover operator: for pairwise matched individuals main and side ingredient subtrees are exchanged.

2.2.2 SmartChef

SmartChef [6] [7] is another approach to generate new cooking recipes using an Evolutionary Algorithm, a general overview is shown in Figure 2.4. SmartChef also uses recipes from popular cooking sites to generate an initial population, represented as recipe trees. In comparison to EvoChef, where each cooking instruction is a node containing a unique identifier referencing to the textual representation of the instruction, cooking instructions are also broken down to one or several tree nodes in SmartChef. Using ingredient information crawled from several databases SmartChef mutates individuals and measures the fitness of the individuals' trees. Crossover is implemented too, which leads to more exploration and the creation of unseens recipes.

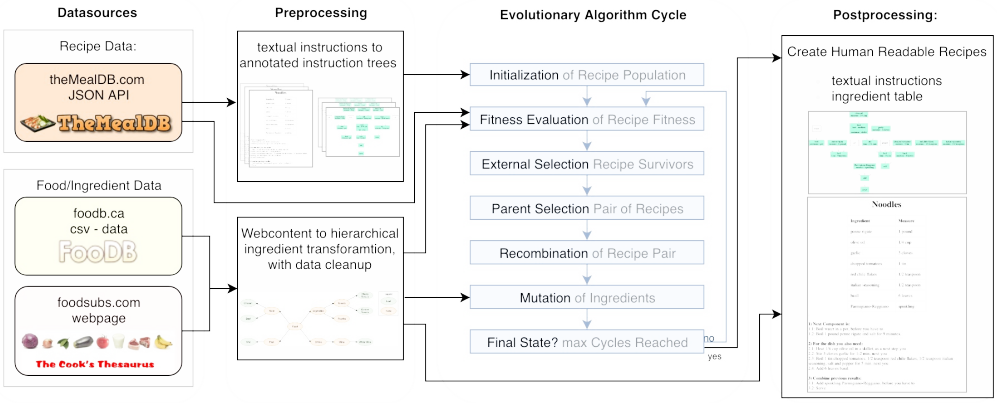


Figure 2.4: Smart Chef Overview, Image from [7]

Chapter 3

Datasets

3.1 Data Representation of Cooking Recipes

There are several ways how recipe data is represented in databases and inside programs that are working with it. Also there are differences what information is given in a recipe depending on the recipe's source. There are datasets in which a recipe only consists of ingredients but other ones contain cooking instructions, ingredient's quantities as well as cuisine style categories. This work focuses on ingredients and cooking actions and learning the relations between them, but Autochef's method could also be extended to cover food containers and quantities in future work (see chapter 6.2). The introduced recipe representations are focused on cooking action and ingredients.

As discussed in this chapter one way to represent recipe data can be a list of textual items and sentences. This representation is easy for humans to read but hard to work with inside an algorithm. Therefore recipe data will be represented in two more formats. The first one focuses on the relations between ingredients and cooking actions and between ingredients itself and stores this relations in adjacency matrices which is discussed in detail in chapter 4. The Evolutionary Algorithm uses a tree structure, similar as it can be used in Genetic Programming [8] which has the advantage that it's easy to modify and to exchange parts of a recipes between individuals. It is furthermore a non linear and hierarchical representation which allows to model parallel cooking procedures but at the same time it is possible to convert the representation back to a serial, human readable cooking recipe as shown in chapter 5.5.

In the following the datasets used in this approach are presented.

3.2 One Million Recipe Dataset

The One Million Recipe Dataset [9] contains one million recipes in English language that are collected from various popular cooking websites. Each dataset's

item contains a title, an ingredient list, an instruction list and an image of the final product. The recipes are provided in plain text as a json array (see listing 1)

```
{
"ingredients": [
{
  "text": "8 tomatoes, quartered"
},
{
  "text": "..."
}
],
"url": "http://www.foodnetwork.com/recipes/gazpacho1.html",
"partition": "train",
"title": "Gazpacho",
"id": "000035f7ed",
"instructions": [
{
  "text": "Add the tomatoes to a food processor
  with a pinch of salt and puree until smooth."
},
{
  "text": "..."
}
]
}
```

Listing 1: Example json object from 1 Million Recipe Dataset [9]

This dataset is used in Autochef as it is one of the biggest recipe datasets that are freely available for research projects. However, the ingredient list is still represented as a set of sentences containing quantities and preprocessing steps (e.g. "quatered", Listing 1) along with the pure ingredients, so that ingredients have to be detected before they can be used.

3.3 Yummly Dataset

AutoChef also uses a dataset that is provided by the cooking website Yummly [5] which contains the ingredient list of 39,774 Recipes annotated with the regional cuisine style (see listing 2)

```
{
"id": 10259,
"cuisine": "greek",
"ingredients": [
    "romaine lettuce",
    "black olives",
    "grape tomatoes",
    "garlic",
    "pepper",
    "purple onion",
    "seasoning",
    "garbanzo beans",
    "feta cheese crumbles"
]
}
```

Listing 2: Example json object from the Yummly Dataset [5]

In comparison to the 1 Million Recipe Dataset (chapter 3.2) the ingredient items here are cleared from other information. Because of this AutoChef uses this dataset to build an ingredient list that is used for ingredient detection together with a self trained classifier (chapter 4.1).

Chapter 4

Recipe Analysis and Information Extraction

The overall goal of the analysis process is to gain information of ingredients and cooking actions and their relations to each other so that this information can be used to build a score function for recipe trees, determine possible mutations of ingredients and cooking actions and generate initial populations (see also chapter 5.1, 5.2 and 5.4). Therefore it is necessary to detect which ingredients are combined and which cooking actions are applied on which ingredients in the 1M recipe dataset [9]. As a preparation step the JSON formatted dataset is converted to a SQL database so that it is possible to access entries and perform queries in a reasonable amount of time.

Finally the extracted information is presented in multiple adjacency matrices with the following properties:

- Matrix $A * I$: the matrix that contains the relations between actions and ingredients.
- Matrix $I * I$: symmetrical matrix that contains the ingredients that are mixed together. This helps, for instance, in deciding that raw ingredients ("beef") are not mixed with cooked ingredients ("boiled rice").
- Matrix $A * I_{base}$: like $A * I$ but only ingredients without their state (so instead of "noodle" and "boiled noodle" there only exists an entry for "noodle").
- Matrix $I_{base} * I_{base}$: like $I * I$ but only ingredients without their state.

The steps that are necessary to create those matrices are now described in detail.

4.1 Extracting cooking related vocabulary

Before it is possible to detect the relations between ingredients and actions, it is necessary to detect ingredients and cooking actions first. An initial list of ingredients is the set of all ingredients that are occurring in the Yummly dataset [5]. Now it is possible to detect some single entities in the ingredient and instruction lists of the 1M recipe dataset [9] that are also appearing in the Yummly dataset [5]. For cooking actions a hand crafted list of verbs is used (see appendix A). Because it is also necessary to detect unseen ingredients a Conditional Random Fields classifier (CRF) [10] is trained on the set of instruction sentences, where known ingredients already are labeled. Conditional random Fields are working with probabilistic graphical models, capable of modelling dependencies between single features, and are therefore well suited for labeling sequence data like sentences written in natural language as shown for example in [11]. As input vectors for the classifier the following features for the word itself and the surrounding ones are used:

- word itself (only for surrounding words)
- POS (part of speech) Tag
- position in sentence
- last two letters
- last three letters

4.1.1 Classifier implementation

Since manually labeling a large amount of recipes is out of scope for this thesis, this approach sticks with the extracted Yummly [5] ingredient vocabulary set and the hand crafted action vocabulary set (appendix A) to label a large amount of recipes automatically. An example of a labeled sentence can be seen in Figure 4.1, represented in the CoNLL-U format [12]. Using this standardized format it is possible to process the data using existing Python packages [13].

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
1	Rinse	rinse	VERB	VB	_	_	_	_	food_type=action
2	the	the	DET	DT	_	_	_	_	_
3	pasta	pasta	NOUN	NN	_	_	_	_	food_type=ingredient
4	in	in	ADP	IN	_	_	_	_	_
5	cold_water	cold_water	NOUN	NN	_	_	_	_	food_type=ingredient
6	and	and	CONJ	CC	_	_	_	_	_
7	set	set	VERB	VB	_	_	_	_	food_type=action
8	aside	aside	ADV	RB	_	_	_	_	_
9	.	.	.	.	_	_	_	_	_

Figure 4.1: Example of an annotated Sentence in CONLLU format. The empty columns are defined by the standard but not used in this thesis.

The used columns are:

- **ID**: Token Position in Sentence
- **LEMMA**: Generalized token, that does not use a specific word ending.
- **UPOS**: Universal Part of Speech Tag
- **XPOS**: Shortend UPOS Tag
- **MISC**: space for user defined additional features. In this approach it is used to add an extra field called *food_type* that can be *action*, *ingredient* or nothing

The NLTK toolkit [14] is used in AutoChef to generate the fields ID, LEMMA, UPOS, XPOS and the additional MISC annotation.

To measure the performance of the CRF classifier 100 sentences (around 730 single tokens) are manually evaluated. In Figure 4.2 shows the binary confusion matrices for each class. The accuracy of both classes (True Positives and True Negatives) is over 90%, but especially the relatively high false positive rate for the action label (compared to ingredient's false positives) causes problems in the recipe analysis process, resulting in ingredients being matched to unreasonable cooking actions. This and the fact that the number of possible cooking actions is much more limited compared to the possible number of ingredients this approach sticks with a curated list of cooking actions (see appendix A) and using the classifier to detect ingredients during the recipe scanning process.

		True Action Label	
		TRUE	FALSE
Predicted Action Label	TRUE	12,60%	2,19%
	FALSE	3,29%	81,92%

		True Ingredient Label	
		TRUE	FALSE
Predicted Ingredient Label	TRUE	10,27%	0,82%
	FALSE	2,60%	86,30%

Figure 4.2: Binary Confusion Matrices for Action and Ingredient Labels

4.2 Recipe Analysis

In the following it is shown how information from Recipes of the 1M recipe dataset [9] is extracted.

4.2.1 Recipe Structure

As a preprocessing step the raw JSON stream of recipes in the 1 Million Recipes dataset is converted to a SQL Library so that it is possible to access single recipe

items in a reasonable time. Each recipe from the 1 Million Recipes Dataset has a title, an ID, a list of ingredients and and a list of instructions (figure 4.3).

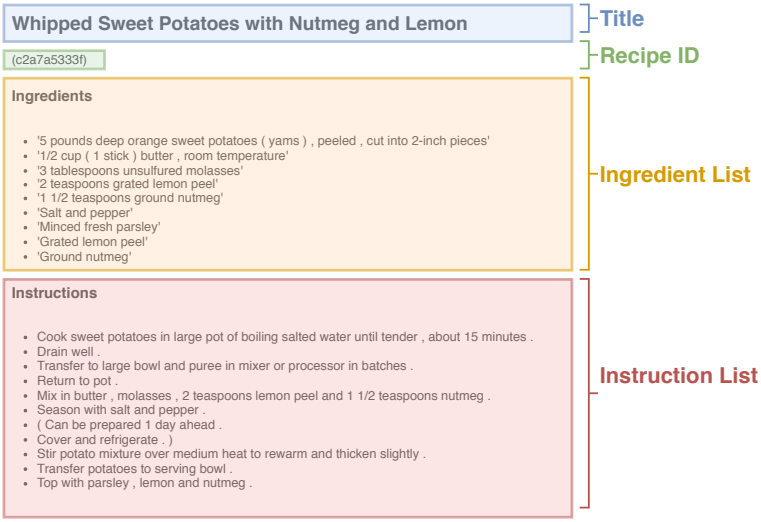


Figure 4.3: An Example Recipe

4.2.2 Process Recipe States and Ingredient States

The term *"state"* is used from now on several times for recipes and ingredients: an ingredient state contains the ingredient itself, cooking actions that are applied on it and a list of other ingredients (and their states) that are mixed together with it. A recipe state consists of a list of several ingredients, including their states. A recipe state can also be modified by a cooking instruction.

At the beginning of a single recipe's analysis AutoChef builds the initial recipe state s_0. For this it scans the recipe's ingredient list to get an initial list of ingredients. As an additional step it detects whether cooking actions have already been applied on ingredients and modifies the ingredient's state in this case (e.g. for the ingredient entry *"cut onion"* the ingredient onion is created and it's state is defined by the action set {cut}).

Then it iteratively processes the instruction set. For each instruction, it detects actions and ingredients and matches them together. The next cooking action left to an ingredient is paired with it. Some exceptions are made for edge cases: if ingredients remain unmatched, it looks also whether cooking actions are next to them to the right. If so, they are matched together. Also, if there is an instruction consisting only of one cooking action without any ingredient it is assumed that this action is applied on the result of the previous step.

Although this is not a very advanced heuristic it works well on most recipes since the instructions are already broken down into little steps and written in imperative English sentences, which are mostly used for giving instructions, commonly have the clean structure that matches the rules described above. However, since

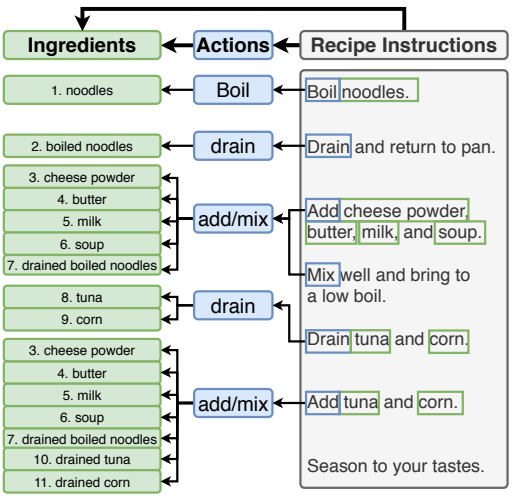


Figure 4.4: Example how information is automatically extracted from a single recipe in AutoChef. After each instruction there exists a new recipe state s_i

the recipes are produced by humans who not had in mind that some program will parse them and expects the recipes to be clearly structured, not all instructions are always written this way. So it is possible that there are wrong matches, that need to be filtered out later in the process (see also section 4.2.3).

The detected ingredient-action pairs are then used to transfer the old recipe state s_{i-1} to the next recipe state s_i which consists of all ingredients and their action sets after the i-th instruction. AutoChef first differs between regular cooking actions and actions that mix several ingredients together. Then it matches the detected ingredients in the i-th instruction to ingredients in the state s_{i-1}. If the detected ingredient has an action matched to it either the corresponding action is added to the ingredient's cooking action set if it's a regular cooking action or it is mixed together with other ingredients in the instruction if it's a mixing action. The result is the new state s_i. An example of this process can be seen in figure 4.4. Finally after the last instruction the process ends in a final state s_n. Based on this state two adjacency matrices are built: the matrix $A * I$ (figure 4.5) contains the detected relations between cooking actions and ingredients (including their state) and the matrix $I * I$ (figure 4.6) represents the detected ingredient's pairwise mixes (including their states).

4.2.3 Postprocessing of adjacency matrices

After merging all adjacency matrices AutoChef applies a threshold on all matrix values to make sure only relations survive that occur multiple times. In this thesis' implementation this threshold is 10 for $A*I$ and $I*I$ and 20 for $A_{base}*I_{base}$ and $I_{base}*I_{base}$ for 100000 random analyzed recipes of the 1 Million Recipe pool. For $A * I$ and $A * I_{base}$ also another matrix is build that is based on cooking action groups ("heat", "prepare" and "cool") instead of specific actions, so that

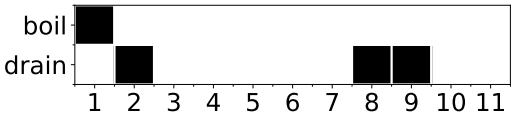


Figure 4.5: Action adjacency matrix $A * I$ for figure 4.4

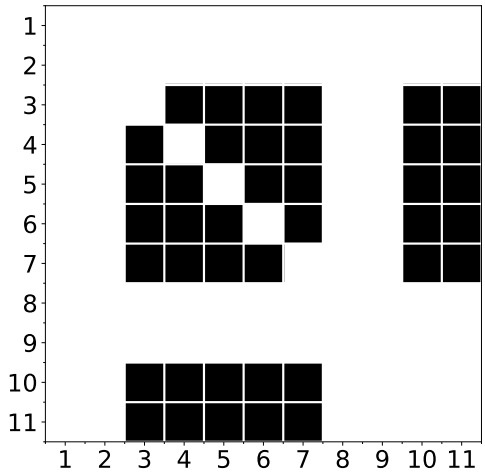


Figure 4.6: Mixing adjacency matrix $I * I$ for figure 4.4

it can be measured if some ingredients are commonly "heated" or "prepared". These matrices are denoted as $A_{grp} * I$ and $A_{grp} * I_{base}$. This information is also used in the fitness evaluation (chapter 5.4).

4.3 Discussion of Data Extraction Results

4.3.1 Pairwise ingredient mixes

The extraction results are stored in adjacency matrices. For visualization of the matrices a graph is built, with the entries as edge weights. The result for the most used ingredients is shown in Figure 4.7. The size of the nodes represents the relative number of occurrences in the dataset and the edge sizes represent the edge weights which correspond to the absolute number of seen pairwise ingredient's mixes. The node positions are determined by a spring force model [15] over the whole dataset, combined with a purely cosmetic position correction to ensure that there is no text collision. The full graph is available in this work's source files. For comparison, the absolute number of occurrences in pairwise ingredient mixes can be seen in Figure 4.8

It can be seen that salt and sugar are the most used ingredients and they are highly connected in the graph. Also egg, flour and butter are closely connected to each other, which may indicates a high number of baking recipes in our dataset. Another observation is that there exist two entities for garlic: *garlic clove* and *clove garlic*. Although they do not have an edge in common, because they never appeared together in a single recipe, they have a similar connectivity so that

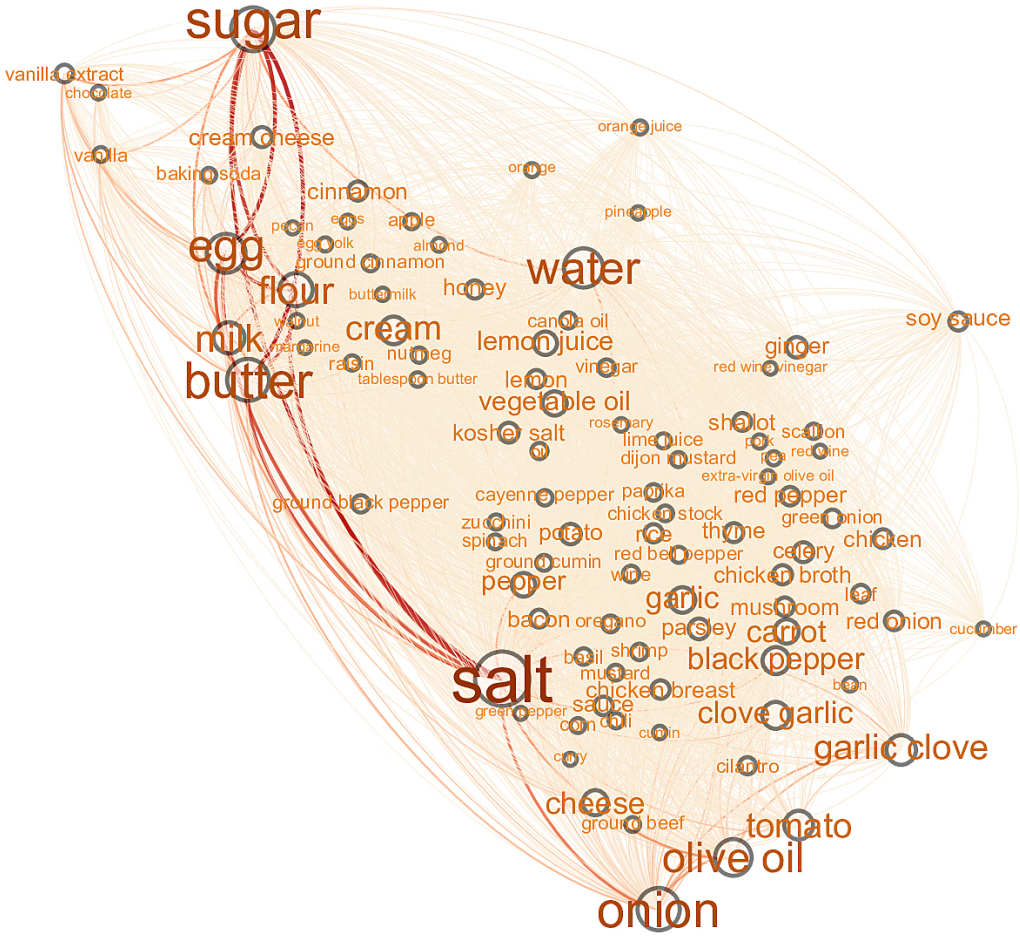


Figure 4.7: Visualized graph for Matrix $I_{base} * I_{base}$, reduced to nodes n with $grad(n) \geq 100$ (Visualization done with Gephi [16])

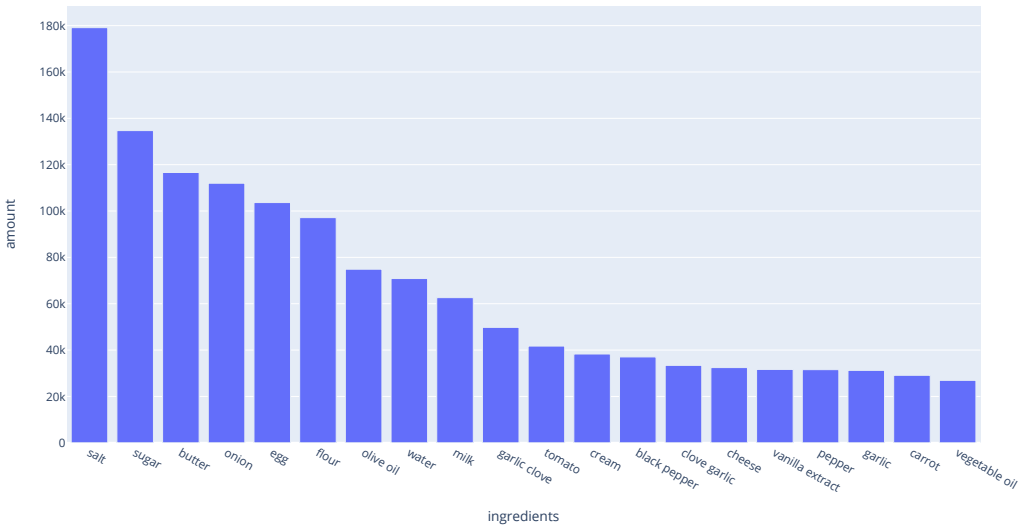


Figure 4.8: Absolute number of the top 20 ingredient occurrences in $I_{base} * I_{base}$

the spring force model forced them to be next to each other. As mentioned in chapter 6.2, future work may include the task of entity recognition in the graphs to resolve doubled entries.

4.3.2 Combinations of actions with ingredients

In figure 4.9 a bipartite graph visualizing all entries in $A_{base} * I_{base}$ with a value higher than 1000 (which means that each action ingredient combination that is visualized edge have been seen at least 1000 times) is shown. The size of the entries represents their node's degree. In addition figure 4.10 displays the cumulative occurrences of all actions in $A_{base} * I_{base}$.

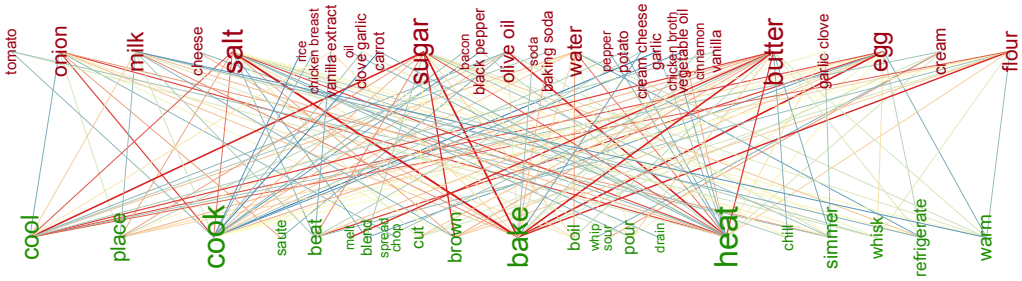


Figure 4.9: Visualized bipartite graph for matrix $A_{base} * I_{base}$, reduced to edges with a weight of at least 1000. Green entries are cooking actions, the red ones ingredients. Edge scheme goes from blue, low weight, to red, high weight. (Visualization done with Gephi [16])

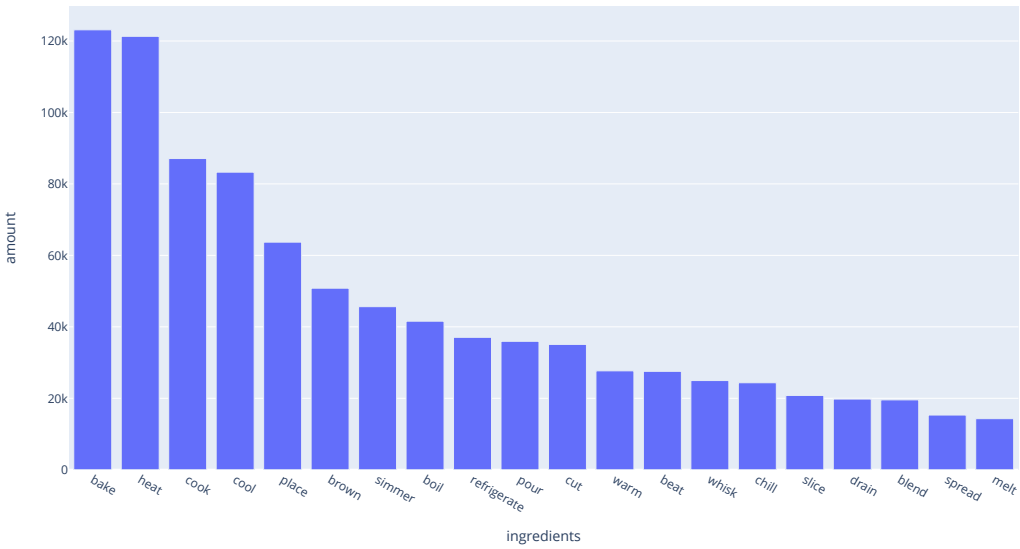


Figure 4.10: Absolute number of all action occurrences in $A_{base} * I_{base}$

It can be seen that *bake* is the most used cooking action followed by *heat* and *cook*. Also that *salt*, *sugar*, *butter*, *egg* and *flour* are the most used ingredients in baking actions but other ingredients like onions are more likely to be *cooked* or

heated. Other ingredients like *butter* have a large variety of applicable cooking actions, it is even as likely to be cooled as it is to be heated. But there are also hints for some ambiguity about cooking vocabulary. So is the edge between *brown* and *sugar* likely to be caused by misinterpretation of the ingredient *brown sugar* to be a combination of the valid cooking action *brown* and the ingredient *sugar*. This could be resolved by improvements of the entity recognition in future work.

Chapter 5

Evolutionary Algorithm

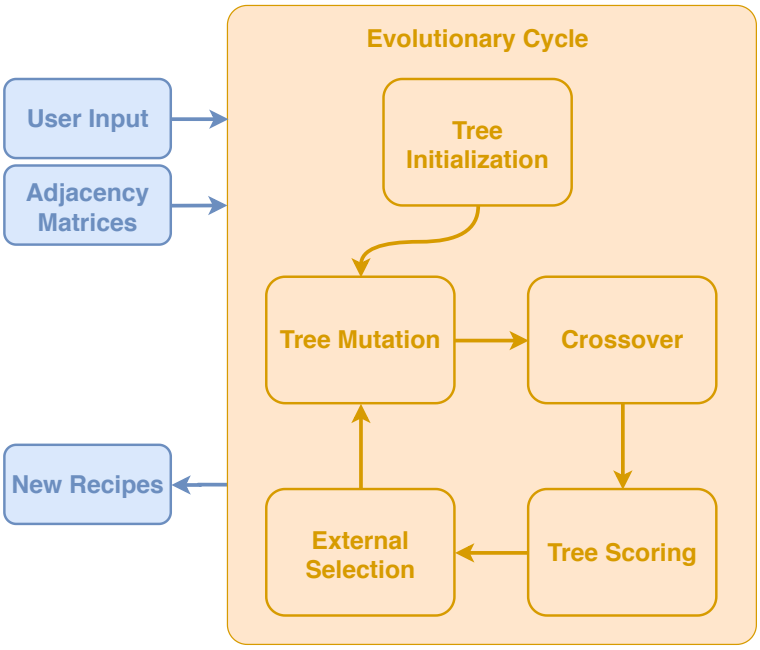


Figure 5.1: Overview of the Evolutionary Cycle

In AutoChef each recipe is represented as a tree. As already mentioned in chapter 3.1 this allows us to represent the cooking procedure in a parallel and non-serial way, which makes it easy to interchange sub-procedures (represented as subtrees) between individuals and implement mutations that respect the hierarchical structure. Also it is easy to transform the recipe back into a serial, textual representation that is understandable for humans as described in chapter 5.5.

A recipe tree consists of several nodes which are connected by directed edges. In this representation the edge directions go from top to bottom, which represents

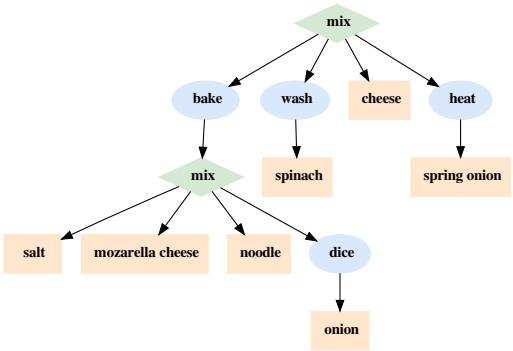


Figure 5.2: example tree

the way trees are traversed, beginning at the root node. This is the opposite way of how cooking instructions are processed in reality since recipes usually start at the ingredients, process and combine them, process and mix the results further until it gets to the tree's root node that represents the final recipe. The trees in this approach have three different types of nodes:

- Ingredient Nodes (yellow rectangle): Ingredient Nodes are the leaf nodes or terminal nodes in the recipe tree. They just store the used ingredients, before any cooking action is applied to them
- Action Nodes (blue ellipse): Action Nodes are function nodes that represent cooking actions that are applied on their subtree. Actions are further separated into different groups: preparation actions (e.g. "cut" or "peel"), heating actions (e.g. "cook") and cooling actions (e.g. "refrigerate")
- Mix Nodes (green diamond shape): Mix Nodes are another type of function nodes that have more that one child. They represent the action of mixing the results of their subtrees together.

An example tree is shown in figure 5.2

5.1 Initialization

In order to generate the initial population of recipe trees, the user defines a set of ingredients (with at least one entry). At least one of them has to marked as "main ingredient". Given the user input, additional ingredients are chosen that are likely to be mixed together with this initial ingredient list.

Using the adjacency matrix $I_{base} * I_{base}$, AutoChef selects the n most mixed ingredients for each ingredient i, creating an list of possible additional ingredients. Later it merges the lists for each ingredient i to a list of action candidates, denoted as C and keeps count of the number of occurrences $s(c)$ in for each

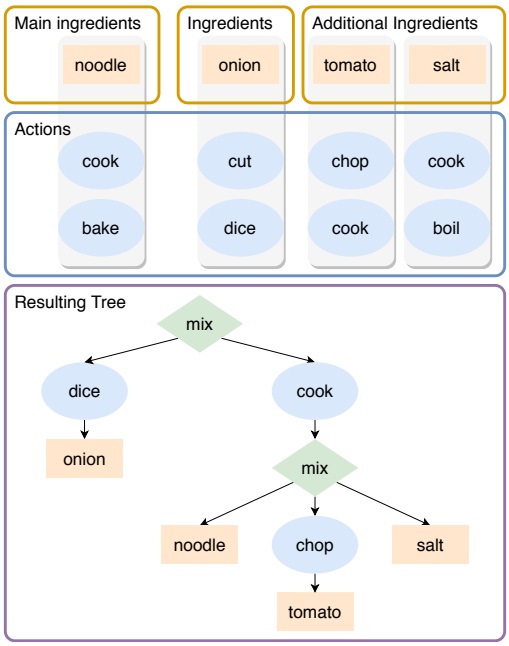


Figure 5.3: Initialization Example. *Main Ingredients* and *Ingredients* are chosen by the user for the whole population, Additional Ingredients and Cooking Actions are generated independently for each individual

ingredient candidate $c \in C$.

The inverse rank $r(c)$ is built for all candidates, with $r(c) = 1$ for the candidate $c \in C$ with the lowest sum $s(c)$ and $r(c) = |C|$ for the candidate $c \in C$ with the highest sum. For each ingredient, AutoChef designs the probability to be chosen as additional ingredient as

$$p(c) = \frac{r(c)}{\frac{1}{2} \cdot |C| \cdot (|C| + 1)} \tag{5.1}$$

This represents a "roulette wheel" like distribution for the selection probabilities, which is also often used as selection strategy in evolutionary algorithms [17]. Its advantage is, that one overly represented entry does not outweighs the other while at the same time give higher entries a higher probability being chosen. An example of the distribution can be seen in figure 5.4.

Depending on the parameter (given by the user) n_+ AutoChef chooses n_+ distinct additional ingredients by the probabilities given with p from the set of given ingredients.

The ingredients are grouped together by compatible cooking actions from the set of "heating" actions. Furthermore for each single ingredient it is looked whether it should be connected to a "preparing" action first. To do this, AutoChef uses the action group adjacency matrix $A_{grp} * I_{base}$ to measure the likelihood of an

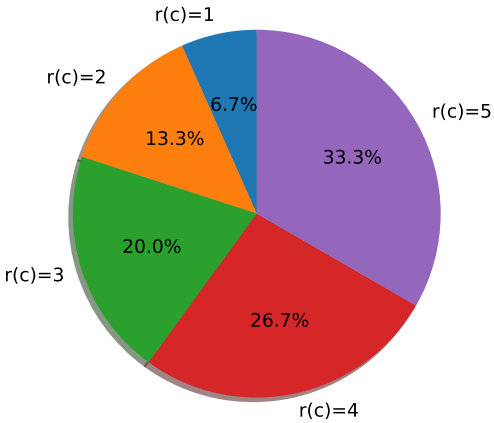


Figure 5.4: Example of the distribution $p(c)$ with 5 ranked items ($|C| = 5$)

ingredient preparation and combines it with a small gaussian distribution to add some randomness in the process. The preparation function is:

$$f_{prep}(i) = \begin{cases} 0, & if\ \frac{p(i)}{h(i)} + \chi < t_p \\ 1, & else \end{cases} \tag{5.2}$$

where i denotes the ingredient, $p(i)$ is the number of seen preparation actions on the ingredient and $h(i)$ the number of seen heating actions. $\chi \sim \mathcal{N}(0, \sigma^2)$ is a randomly chosen value with a small variance. And t_p is a threshold for the ratio, set to 0.35 in our experiments based on observed values for $p(i)$ and $h(i)$. Value $\frac{p(i)}{h(i)}$ measures the ratio between heating and preparing actions. Cooling actions are not taken into account here, because (in our case) the number of cooling actions is very small and it would not affect the ratio substantially. Similarly, a function to select a heating action is defined as:

$$f_{heat}(i) = \begin{cases} 0, & if\ 1 - \frac{p(i)}{h(i)} + \chi < t_h \\ 1, & else \end{cases} \tag{5.3}$$

with $t_h = 0.65$.

If $f_{prep}(i) = 1$ for an ingredient i AutoChef chooses a valid preparation action by using the adjacency matrix $A * I_{base}$ (and filtering out all actions that are not a preparation action) and a rank based probability distribution as seen in equation 5.1. After that it randomly chooses matching heating actions in the same way. If some ingredients have the same chosen actions in common, they are mixed together first, then the action is applied on them. If the result is not a single tree yet, all existing subtrees are merged together with a mix node as final step.

5.2 Mutation

The purpose of the mutation step is to modify the tree structure in order to explore new cooking mechanics. In general there are two different elements that AutoChef can mutate: the nodes itself or their edges.

The edge mutation is always possible on every node and is the insertion of a new Action Node, displayed as example in figure 5.5

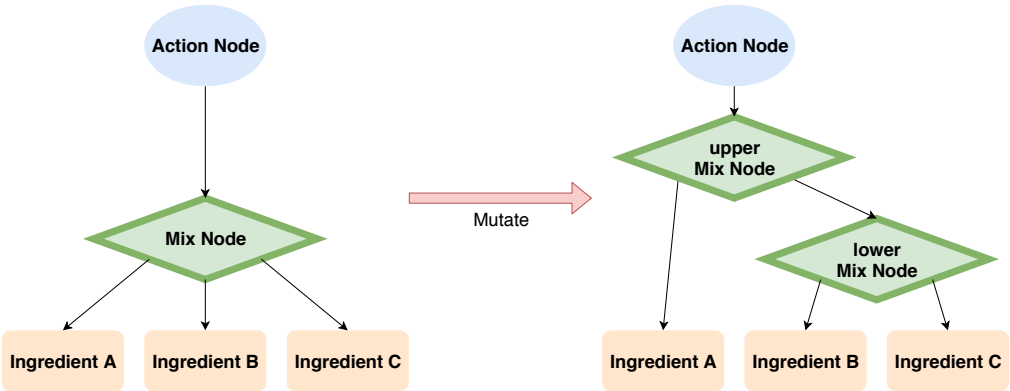


Figure 5.5: Example insertion of an Action Node before another Action Node (this mutation is possible on every node type)

The mutation of the node itself differs for every node type and is shown below.

In order to restrict the exploration of the algorithm (e.g. there is an initial set of ingredients that must be in the final recipe tree) every node can be set to be a constant. Mutations are only applied on non-constant nodes.

5.2.1 Mutation of Mix Nodes

If a Mix Node mutates, AutoChef separates the Mix Node (n_{old}) into a "lower" (n_l) and an "upper" (n_u) Mix Node. Then it chooses a subset of the old node's children (C_{old}) randomly to build the child set of the "lower" node ($C_{n_{\text{l}}} \subset C_{n_{\text{old}}}$). The children of the upper node are defined as the remaining set of children ($C_{n_{\text{u}}} = (C_{n_{\text{old}}} \setminus C_{n_{\text{l}}}) \cup \{n_l\}$). See also Figure 5.6 for an example.

5.2.2 Mutation of Action Nodes

To mutate the Action Node, AutoChef either deletes it (as displayed in figure 5.7) or replaces it by a random action.

Note that the removal of the Action Node in the delete mutation may result in two adjacent Mix Nodes. To avoid tree fragmentation with many Mix Nodes, they are merged together in an additional step.

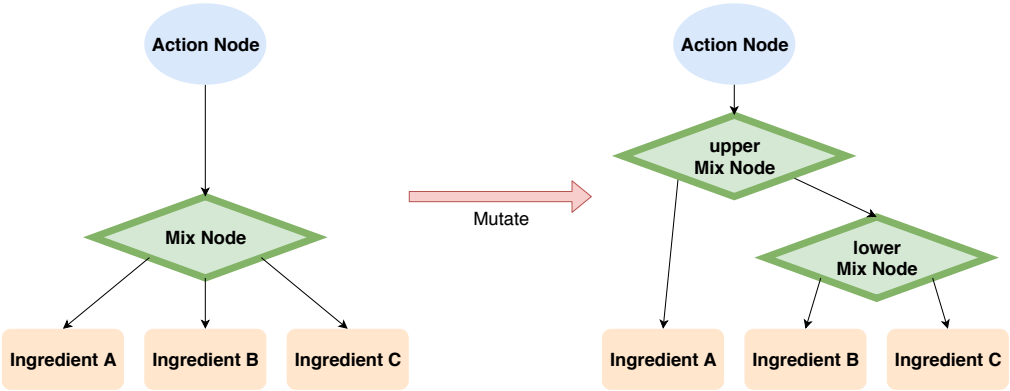


Figure 5.6: example mutation of a Mix Node

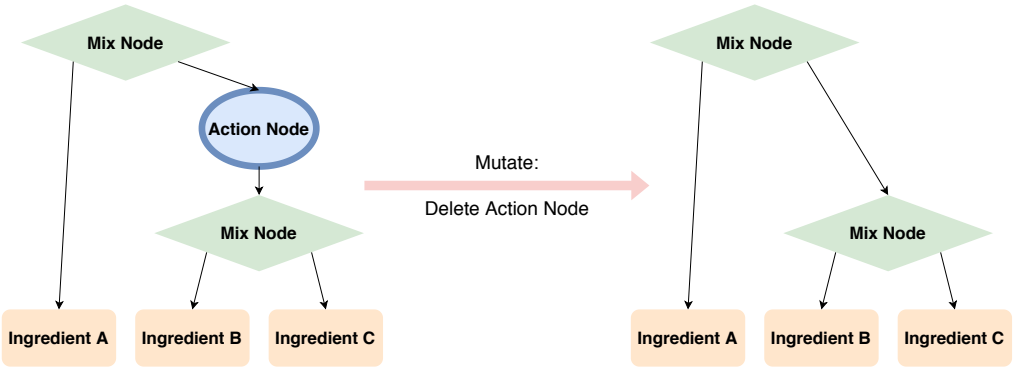


Figure 5.7: Delete Action Node

5.2.3 Mutation of Ingredient Nodes

There are two ways to mutate an Ingredient Node. At first AutoChef looks if the ingredient to mutate is present in a food substitution graph, generated by the SmartChef [7] approach. An example of the graph is displayed in figure 5.8.

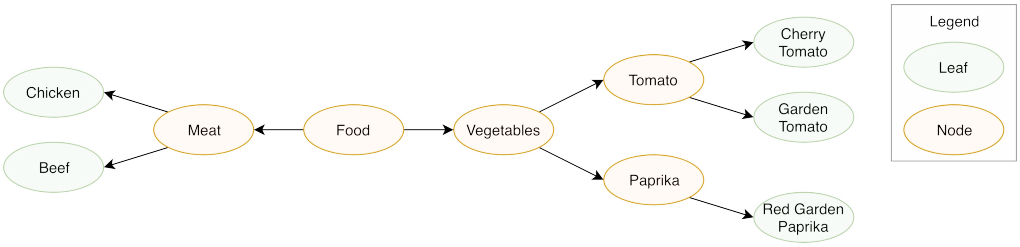


Figure 5.8: Foodsub graph example, Image from SmartChef [7]

If the ingredient is present there it is replaced with a sister ingredient. If not AutoChef takes a look at the ingredient mixing matrix $I * I$ and creates a set of ingredients that are most likely to be mixed with the ingredient to mutate. From this set it then chooses a random ingredient by the same probability distribution shown in equation 5.1.

5.3 Crossover

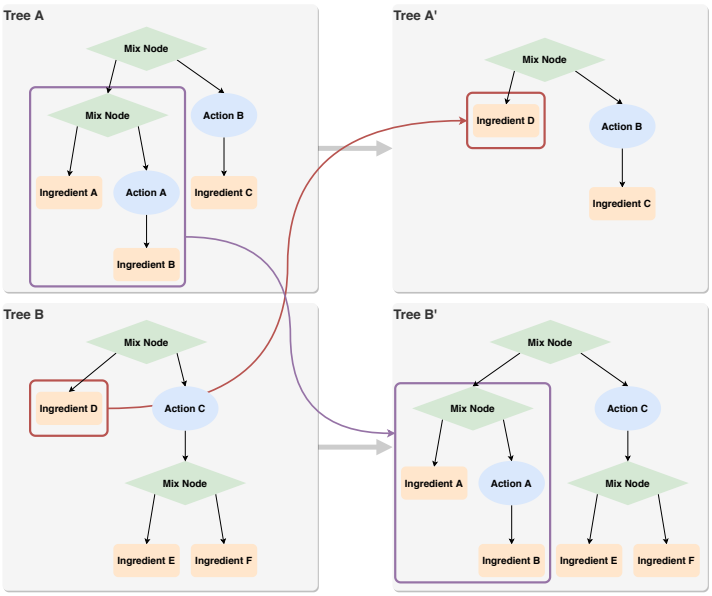


Figure 5.9: Example crossover between two trees

As described in chapter 2.1 crossover is a key tool in evolutionary algorithms to increase the exploration of the search space by combining elements of fit genomes.

In AutoChef crossover is implemented by an exchange of subtrees between two individuals. First all individuals are matched randomly pairwise together and they exchange a random subtree between each other. One example crossover is shown in Figure 5.9.

5.4 Fitness Evaluation

Finally the population's recipe trees have to be evaluated to perform an external selection. There are two strategies available for an external selection based on the fitness scores. In the first one AutoChef randomly groups all individuals from the new and old population pairwise together and measures the tree scores. The individual with the best score survives. In the second one just the best n individuals survive, with n as the defined population size. Which strategy is used can be setup in AutoChef. To measure the fitness AutoChef uses the adjacency matrices from chapter 4.2 to build confidence values for each node. In the following it is first discussed how a score for every single node is built and then, how they can be used to build a score for a whole recipe tree.

5.4.1 Ingredient Node Scores

Ingredient Nodes have 3 scores that are combined together. The binary heat score $s_{heat}(i)$ for the ingredient i is defined similar to f_{heat} (equation 5.3), used in the initialization process.

$$s_{heat}(i, b_h) = \begin{cases} 0, & if\ 1 - \frac{p(i)}{h(i)} > t_h + \epsilon \wedge b_h = 0 \\ 0, & if\ 1 - \frac{p(i)}{h(i)} < t_h - \epsilon \wedge b_h = 1 \\ 1, & else \end{cases} \tag{5.4}$$

$b_h \in \{0, 1\}$ is a binary value, indicating whether i is heated in the recipe tree or not. $2 \times \epsilon$ defines a corridor around the threshold t in which wrongly unheated or heated ingredients are not punished. Similar to s_{heat}, s_{prep} is defined as:

$$s_{prep}(i, b_p) = \begin{cases} 0, & if\ \frac{p(i)}{h(i)} > t_p + \epsilon \wedge b_p = 0 \\ 0, & if\ \frac{p(i)}{h(i)} < t_p - \epsilon \wedge b_p = 1 \\ 1, & else \end{cases} \tag{5.5}$$

with $b_p \in \{0, 1\}$ indicating whether i is prepared. The score s_a punishes duplicate actions in the tree.

$$s_a(i) = \frac{n_a(i) - n_{da}(i)}{n_a(i)} \tag{5.6}$$

where $n_a(i)$ defines the number of actions applied on i and $n_{da}(i)$ defines the number of duplicate actions applied on i.

The final Ingredient Node score is then defined by the normalized sum of the scores s_{prep} and s_{heat}, weighted with s_a (since this is a harder punishment than using it in the normalized sum)

$$s_{ing}(i) = \frac{1}{2} \cdot \big(s_{prep}(i) + s_{heat}(i)\big) \cdot s_a(i) \tag{5.7}$$

5.4.2 Action Node Scores

For the Action Node score $s_{act}(a)$ AutoChef first has to traverse the ingredients in the subtrees of the action node (set I_a). For each traversed ingredient $i \in I_a$ it applies all actions upon to the current action node a on it. Then it takes a look in the adjacency matrix $A * I$ whether a is a valid action on the ingredient i. If so it sets $s_i = 1$, otherwise $s_i = 0$. Then the score is

$$s_{act}(a) = \frac{1}{|I_a|} \sum_{i \in I_a} s_i \tag{5.8}$$

5.4.3 Mix Node Scores

Similar to the Action Node score the Mix Node score is the normalized sum of binary values. First AutoChef traverses the ingredients of all subtrees for Mix Node m and stores all found ingredients in the set I_m. Then every possible pair i, j between the ingredients in I_m is created and each pair binary rated by a lookup in the mix graph $I * I$:

$$s_{mix}(m) = c_m \cdot \sum_{i \in I_m} \sum_{j \in I_m} s_{ij} \cdot t_{ij} \tag{5.9}$$

with

$$\begin{aligned} t_{ij} &= \begin{cases} 0 & \text{if } i \text{ and } j \text{ in same subtree} \\ 1 & \text{else} \end{cases} \\ s_{ij} &= \begin{cases} 1 & \text{if } i \text{ and } j \text{ are a valid mix} \\ 0 & \text{else} \end{cases} \\ c_m &= \text{number of childs of m} \end{aligned} \tag{5.10}$$

using t_{ij} to make sure that only pairs are rated that are not coming from the same subtree. For those pairs it either holds that $i = j$ or that they are already rated by another Mix Node score in the subtree. s_{ij} is one if it is valid to mix i and j according to $I * I$. The score is weighted by the number of childs c_m.

5.4.4 Tree Score

The final tree score s_{tree} is a combination of all single node scores combined with some weights to rate the overall structure of the tree. The node score is the sum of all single scores in the recipe tree, normalized by the number of action scores, ingredient scores and sum of all mixes:

$$s_{nodes} = \frac{\sum_{i \in I} s_{ing}(i) + \sum_{a \in A} s_{act}(a) + \sum_{m \in M} s_{mix}(m)}{|I| + |A| + \sum_{m \in M} c_m} \tag{5.11}$$

and the final tree score is defined as

$$s_{tree} = (\alpha \cdot s_{nodes} + \beta \cdot s_{root}) \cdot s_{duplicates} \cdot s_{acts} \cdot s_{ings} \tag{5.12}$$

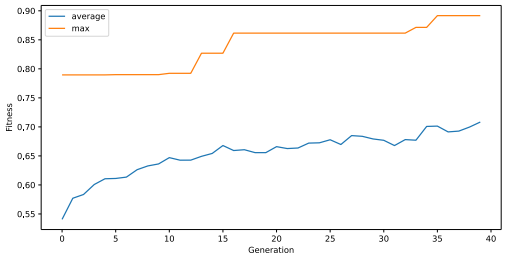
where $s_{acts} = 0$ if the total number of action nodes is below 3 and $s_{ings} = 0$ if the total number of ingredient nodes is below 3. Otherwise both components are set to 1. Also AutoChef punishes the occurrence of duplicate actions:

$$s_{duplicates} = \frac{n_d}{n_a} \tag{5.13}$$

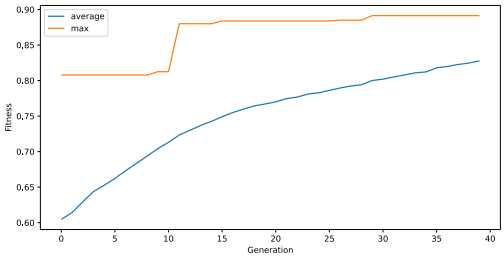
where n_d denotes the number of distinct actions in the tree and n_a the total number of actions.

Another factor is the root node's score s_{root}. Since the root node represents the final recipe it is ensured that this score is more important than a regular node's score. s_{nodes} and s_{root} are linear combined by the weights α and β with $\alpha + \beta = 1$, $\alpha >> \beta$. In the following runs the values are set to $\alpha = 0.9, \beta = 0.1$.

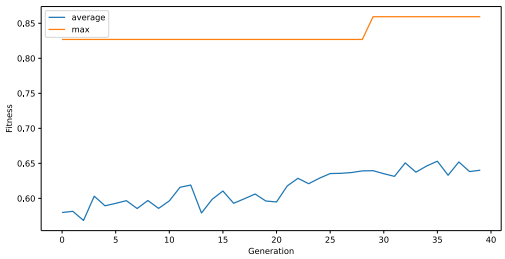
In figure 5.10 the development of a population's average and maximum fitness can be compared between different mutation rates and external selection methods.



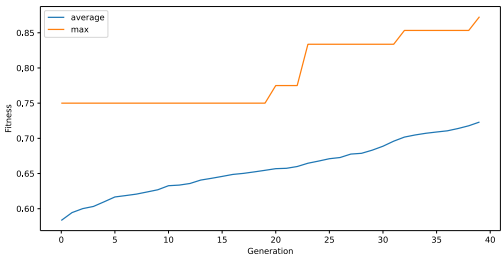
(a) Mutation rate: 1 per tree
external selection: pairwise competition



(b) Mutation rate: 1 per tree
external selection: hold best



(c) Mutation rate: 2 per tree
external selection: pairwise competition



(d) Mutation rate: 2 per tree
external selection: hold best

Figure 5.10: Average and maximum individual's fitness of a population with 75 individuals over 40 generations, generated with user input: "noodle" as main ingredient

It can be observed that a lower mutation rate leads to a faster growing average fitness, however setting the mutation rate too low could lead to a population with very similar individuals which decreases the populations diversity and thus decreases the exploration of the search space. Another observation is that a pairwise competition method can lead to a decreased average fitness in the next generation but the maximum fitness will still always be the same or increasing. Figure 5.11 compares the fitness distributions between the configurations shown in figure 5.10.

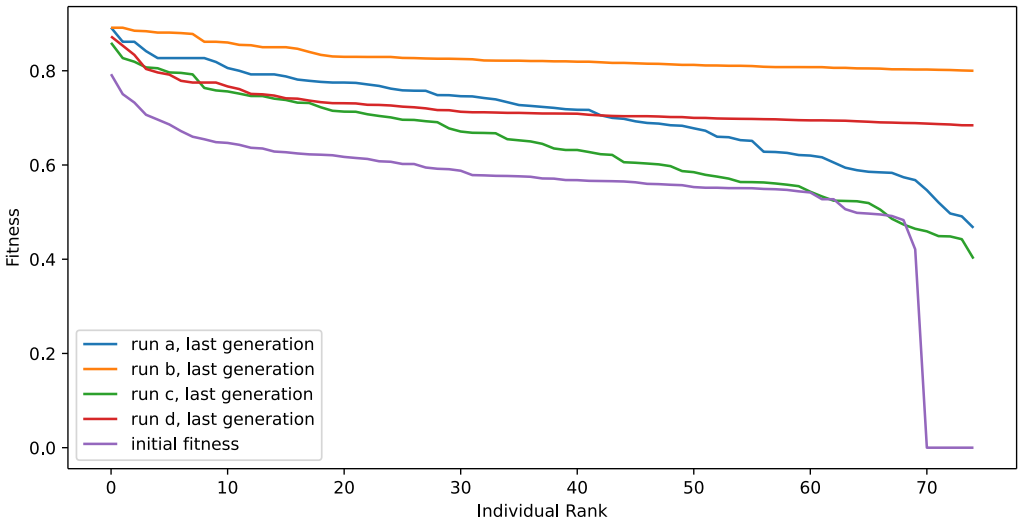


Figure 5.11: Fitness distribution comparison after 40 evolutionary cycles between the configurations displayed in figure 5.10 and the initial fitness

The initial fitness contains some recipes with a fitness value of zero, meaning that they have an invalid recipe structure (e.g. not enough action nodes). Every configuration leads to a higher population's fitness distribution. The configurations using a pairwise competition contain a wider range of fitness values in their last generation, due to the fact that individuals with a bad fitness value have a chance to survive if they are randomly matched together with an individual with an even lower fitness.

5.5 Textual Tree Representation

The resulting recipes trees are converted into corresponding text representation for better human readability. First a depth-first search on the recipe tree is made and an instruction step for every Action Node created. To avoid having a lot of small instructions AutoChef merges instructions of Action Nodes which only have ingredients as children with the text of the instruction of the next layer. Additionally, if the root node is a Mix Node, this Mix Node also generates an instruction set. As example, the textual representation of the tree in figure 5.2 is shown in table 5.1

	ingredients
	spring onion noodle salt spinach mozarella cheese cheese onion

step	**instruction**
1	dice onion and mix it with salt, mozarella cheese and noodle. Then bake it.
2	wash spinach, heat spring onion and mix it with cheese and mix it together with the results of step 1.

Table 5.1: Instructions for recipe tree shown in figure 5.2

Chapter 6

Evaluation and Discussion of Results

6.1 Survey

To measure the performance of the algorithm a survey was created. Each participant got presented four random recipes from a set of 20 recipes generated by AutoChef. The configuration was set as follows:

- **mutation rate:** 1 node in every tree
- **population size**: 50
- **number of evolutionary cycles**: 40
- **external selection**: pairwise tournament selection

For the first 10 recipes the user input was *noodle* for the other ones *rice*, as main ingredient, no other ingredients given. The number of possible additional ingredients, chosen while building the initial population, is set to a minimum of 4 and maximum of 13.

For each presented recipe the user has to answer the following questions:

Binary Questions

- Is this a valid recipe?
- Does it seem eatable?

Questions answered in a range from 0 (negative) to 3 (positive)

- Are the instructions understandable?
- How tasty is it probably?
- Would you cook it?

- How creative is this recipe?
- Is this a good combination of ingredients?
- Are the used cooking actions suitable for the ingredients?

There were 2 surveys made. The fist one was done to collect data for a paper presenting AutoChef and was performed at an early state of the algorithm when only mutation and fitness evaluation but no crossover was implemented. The second one was done with all features, including crossover. The results are presented in Figure 6.1

Recipe property	Yes	No
Validity (1)	88.42%	11.58%
Edibility (2)	89.42%	10.58%

Recipe property	0	1	2	3
Understandable (3)	1.47%	5.88%	26.47%	66.18%
Tasty (6)	2.94%	27.94%	44.12%	25.00%
Would you cook it (7)	17.65%	30.88%	29.41%	22.06%
Creativity (8)	1.47%	29.41%	50.00%	19.12%
Ingredient combination (4)	4.41%	19.12%	35.29%	41.18%
Cooking actions (5)	10.29%	26.47%	41.18%	22.06%

(a) First Survey (without crossover)

Recipe property	Yes	No
Validity (1)	85.83%	14.17%
Edibility (2)	90.00%	10.00%

Recipe property	0	1	2	3
Understandable (3)	0.00%	5.00%	45.00%	50.00%
Tasty (6)	7.50%	37.50%	45.00%	10.00%
Would you cook it (7)	10.00%	42.50%	35.00%	12.50%
Creativity (8)	5.00%	35.00%	40.00%	20.00%
Ingredient combination (4)	5.00%	30.00%	45.00%	20.00%
Cooking actions (5)	0.00%	22.50%	52.50%	25.00%

(b) Second Survey

Figure 6.1: Survey Results

To summarize the results there are two scores. The first one, the correctness score, is the average over all answers of the questions of validity, edibility and ingredient combinations. The second one is the innovation score. It's build by the average of the remaining questions (ingredient combination, tastiness, creativity and whether the participant would cook it). The scores are displayed in Figure 6.2

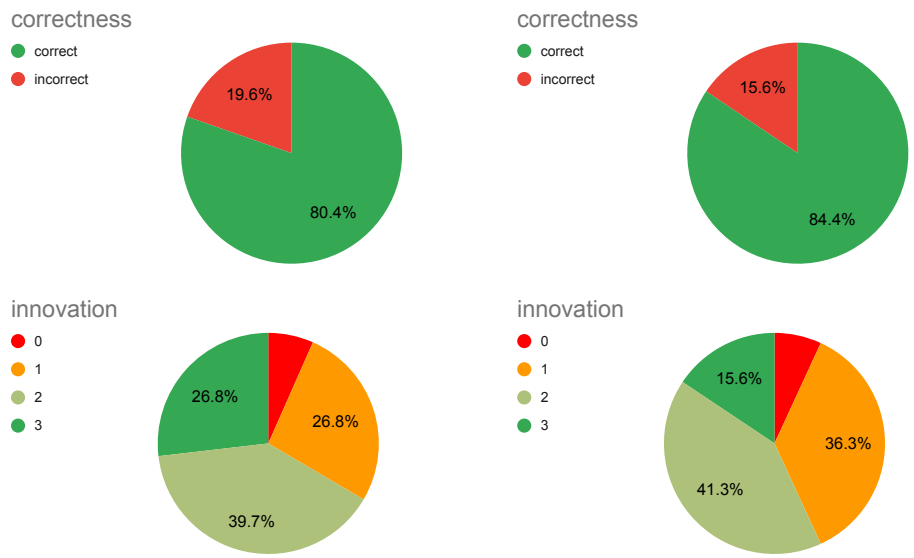
It can be observed that with crossover implemented the user scores are mostly similar, notable differences are that correctness values are slightly increasing and the top values for innovation slightly decreasing.

6.2 Conclusion and Future Work

This thesis has shown that it is possible to learn the usage of ingredients and cooking procedures automatically from large cooking recipe databases and using the information to autonomously generate new cooking recipes that are valid.

Given this approach a user is able to perform a request with ingredients that should be used in a cooking recipe for an initial state and AutoChef generates a new valid recipe in a textual representation that can be used by the user.

Future work may include more domain specific information about ingredients, spices and cooking actions that can be used as additional information in the mutation process and the fitness function.



(a) First Survey scores (without crossover)

(b) Second Survey scores

Figure 6.2: correctness and innovation results

Furthermore the process can be extended to extract ingredient amounts and cooking containers from recipe databases since AutoChef does not have this information in the current state. Including this information it should be possible to create more fine granular cooking steps.

Also the recipe analysis step (see chapter 4.2) can be improved by using more advanced heuristics to match ingredients and actions together. The structure of imperative English sentences makes it easy to create a simple heuristic that works in a lot of cases like it's done in AutoChef, but a more advanced matching could further improve the results of the analysis process.

Furthermore additional metrics and heuristics to validate the created recipes could be included. In this thesis the validity and innovation of the recipes was tested with user surveys. However, although the created recipes are in a human readable textual format, they still differ in their structure from recipes found on cooking sites which makes it hard to compare them directly. Future work may be able to shorten this gap.

Bibliography

[1] *IBM Chef Watson (closed)*. URL: http://www.ibmchefwatson.com.

[2] *Official Python Website*. (last visited: 2020-04-29). URL: https://python.org.

[3] John H. Holland. "Genetic Algorithms". In: *SCIENTIFIC AMERICAN* 267.1 (1992), pp. 66–73.

[4] Hajira Jabeen, Nargis Tahara, and Jens Lehmann. "EvoChef: Show Me What to Cook! Artificial Evolution of Culinary Arts". In: *Computational Intelligence in Music, Sound, Art and Design*. Ed. by Anikó Ekárt, Antonios Liapis, and María Luz Castro Pena. Vol. 11453. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 156–172. ISBN: 978-3-030-16666-3. DOI: 10.1007/978-3-030-16667-0_11. URL: http://link.springer.com/10.1007/978-3-030-16667-0_11.

[5] *Yummly Dataset on kaggle.com*. (last visited: 2020-02-23). Jan. 2017. URL: https://www.kaggle.com/kaggle/recipe-ingredients-dataset.

[6] Carsten Draschner, Jens Lehmann, and Hajira Jabeen. "Smart Chef: Evolving Recipes". In: *EVO** (2019), p. 8.

[7] Carsten Draschner. *'Smart Chef – Evolving Recipes', Master Thesis at 'Smart Data Analytics - Institut für Informatik III', Rheinische Friedrich-Wilhelms-Universität Bonn*.

[8] P. G. Espejo, S. Ventura, and F. Herrera. "A Survey on the Application of Genetic Programming to Classification". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.2 (2010), pp. 121–144.

[9] Javier Marin et al. "Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images". en. In: *arXiv:1810.06553 [cs]* (Oct. 2018). arXiv: 1810.06553. URL: http://arxiv.org/abs/1810.06553.

[10] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. ISBN: 1558607781.

[11] Andrew McCallum and Wei Li. "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons". In: *Proceedings of the seventh conference on Natural language*

learning at HLT-NAACL 2003 -. Vol. 4. Association for Computational Linguistics, 2003, pp. 188–191. DOI: 10.3115/1119176.1119206. URL: http://portal.acm.org/citation.cfm?doid=1119176.1119206.

[12] (last visited: 2020-02-23). URL: https://universaldependencies.org/format.html.

[13] *Conllu Python Package*. (last visited: 2020-04-29). URL: https://github.com/EmilStenstrom/conllu/.

[14] *NLTK Website*. (last visited: 2020-01-26). URL: https://www.nltk.org/.

[15] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph drawing by force-directed placement". In: *Software: Practice and Experience* 21.11 (Nov. 1991), pp. 1129–1164. ISSN: 00380644, 1097024X. DOI: 10.1002/spe.4380211102.

[16] *Official Gephi Website*. (last visited: 2020-04-29). URL: https://gephi.org/.

[17] Jinghui Zhong et al. "Comparison of performance between different selection strategies on simple genetic algorithms". In: *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*. Vol. 2. IEEE. 2005, pp. 1115–1121.

Appendix A

Cooking Action Set

The hand curated list of cooking actions and their ordering to the groups *heat*, *prepare* and *cool*

Action	Group
heat	heat
cook	heat
bake	heat
grill	heat
melt	heat
blend	
beat	prepare
cool	cool
brown	heat
cut	prepare
chill	cool
drain	
boil	heat
simmer	heat
pour	
freeze	cool
saute	heat
rinse	prepare
warm	heat
wash	prepare
knead	prepare

Action	Group
peel	prepare
parboil	heat
break	prepare
broil	heat
scorch	heat
skim	
fry	heat
refrigerate	cool
burn	heat
thicken	
grate	prepare
brush	prepare
open	prepare
crack	prepare
poach	heat
slice	prepare
whisk	
dice	prepare
marinate	
whip	
sour	

Action	Group
soak	
steam	heat
chop	prepare
mince	
mash	prepare
squeeze	prepare
wipe	prepare
thaw	prepare
curdle	heat
sweeten	
baste	
carve	
grind	prepare
debone	prepare
steep	
clarify	
macerate	prepare
crumple	prepare
braise	heat
gut	
bury	

Appendix B

Attached Digital Content

B.1 CDRom Content

Overview of the attached CDRom's root directory. Also available as readme.html on the device itself

B.1.1 Additional data

Here is an overview about the content located in the subdirectories:

- `./AutoChef`: AutoChef Python Package
- `./Presentation`
  - Master Thesis Presentation as .html and .pdf
- `./Plots`
  - Graph Plots (created with gephi)
    - `action_graph.gephi` : $A_{base} * I_{base}$, reduced to most used combinations
    - `ingredient_graph.gephi`: $I_{base} * I_{base}$, reduced to most used ingredient mixings
- `./Surveys`: Raw data collected in user surveys and the survey recipes

B.2 Autochef Python Package

The easiest way to use AutoChef is by the Autochef Package, located in the folder `./Autochef`. In the following it is described how it can be installed in a python environment on a linux host system and how to use the libraries' interface to easily run an evolutionary algorithm with your own parameters.

If you want to work on some intermediate steps, like generating adjacency matrices on new recipe datasets, look at the documentation inside the package folder.

B.2.1 Install AutoChef as Python Package

The easiest way to try out Autochef is by installing the Autochef repository as python package on a linux distribution with Python 3.7 (tested on Ubuntu 19.10).

Note that there is some data prepackaged in a binary format that may is not compatible on other operation systems like macOS or Windows. On this systems the adjacency matrices have to be regenerated manually. Instruction how to do this can be found at the documentation inside the package folder.

Dependencies to install before:

- graphviz
- python3.7
- pip
- around 8 GiB of RAM

on ubuntu 19.10:

```
sudo apt install graphviz graphviz-dev python3-pip
python3 python3-dev python3-venv git
```

Note for Ubuntu 20.04 and python3.8:

Ubuntu 20.04 ships with python3.8 per default. Since there are compatibility issues with packages used for deserialization of data in python3.8 at the point of writing this documentation, either use the package `AutochefPy38` instead of `Autochef`, which is also available in this directory, or recreate the adjacency matrices as described in the package documentation.

Install AutoChef as Python Package

first extract the Autochef Archive. Then install it with

```
pip install ./Autochef
```

the relative path `./Autochef`, containing the file `setup.py`, should be adjusted properly if the command is not run from the directory of the readme file

Notes

- On Ubuntu you may have to use `pip3` instead of `pip`
- If you don't want to install the package system wide you can create a virtual environment first and install then the package like described above:

```
python3 -m venv ./test
source ./test/bin/activate
pip install wheel
pip install -U setuptools pip
pip install ./Autochef
```

B.2.1.1 Example usage

If Autochef is installed correctly you can try out the evolutionary algorithm in an interactive python console (like a jupyter notebook). For example:

Import autochef's Evolutionary Algorithm

```
import autochef
from autochef.EvolutionaryAlgorithm import EvolutionaryAlgorithm as EA
```

Setup parameters

```
# define the set of input ingredients and it's subset of main ingredients

all_ingredients = ["bread", "butter"]
main_ingredients = ["bread"]

# define range of additional ingredients
min_add = 4
max_add = 13

# population size:
n_pop = 50

# mutations rate (given in number of node mutations for each tree)
mutations = 2

# if pairwise competition is false, just the best n_pop/2 individuals survive
pairwise_competition = True
```

build initial population

```
p = EA.Population(
    all_ingredients,
    main_ingredients,
    min_additional=min_add,
    max_additional=max_add,
    n_population = n_pop,
    mutations=mutations,
    pairwise_competition=pairwise_competition
)
```

run over a given number of generations

```
# run over 10 cycles
fitness_logs = p.run(10)
```

plot best 3 individuals

```
p.plot_population(n_best=3)
```