Technical Report CS??-??

Particle Swarm Optimization for Businesses: Bakery Manager

Trevor J. Kroon

Submitted to the Faculty of
The Department of Computer Science

Project Director: Dr. Oliver Bonham-Carter
Second Reader: Dr. Aravind Mohan

Allegheny College
2018

I hereby recognize and pledge to fulfill my responsibilities as defined in the Honor Code, and to maintain the integrity of both myself and the college community as a whole.

Trevor J. Kroon

TREVOR J. KROON. Particle Swarm Optimization for Businesses: Bakery Manager.
(Under the direction of Dr. Oliver Bonham-Carter.)

ABSTRACT

There are bakery businesses that have potential to be successful, but misuse their resources. The purpose of this is to explore the use of Particle Swarm Optimization (PSO), to optimize a menu so that it would benefit the baker. We do this by minimizing tracking the history of clients, and then moving recipes around accordingly.

Contents

List of Figures

Chapter 1

Introduction

There is a problem in this world right now revolving around the use of resources in multiple different functions. The purpose of this system is to figure out a method for bakeries to spend their time baking, while this software system figures out a schedule for the day to make baked goods. This will allow them to minimize the amount of time spent doing arbitrary tasks, and then they can spend time doing what they are passionate about, baking. The system utilizes Combinational Particle Swarm Optimization (CPSO) and relational databases. Because of the nature of CPSO, using relational databases will lower the amount of information it has to query through and increasing its speed

1.0.1 Motivation

Looking on the market, there are several baking software systems that help out the baker in one or two ways. The purpose of this is to research and develop an automated system that solves the problem of management of resources and scheduling. This will be accomplished through the use of constructing Relational Databases to help with optimizing the use of resources by a baker and the associated schedule to effectively execute that plan. This software system will allow for the bakers to not have to focus as much of their time towards scratching out tentative plans and needing to make emergency purchases of ingredients. Although this is constructed for the purpose of a bakery, this could be applied to many different functions, with switching around the wording and the meaning to the user. Additionally this system is being created to be easy for individuals with no prior coding experience. Specifically the features of this program is the management of the inventory of the bakery, what the bakery needs in terms of ingredients, a scheduler that can be interacted with to optimize the use of resources, seasonal restrictions on certain recipes, memory bank of available tools, and logging information regarding the price and amount of revenue generated from a good. Considering the scarcity of resources and competition for resources, this is also an important project for finding optimal solutions. One example would be scheduling a wedding several weeks ahead of time, and sending weekly reminders so that the bakery can prepare what they can in advance, and to make sure they have those needed ingredients on time. This is a point of failure in a bakery, because through having several weddings a week, on top of making everyday pastries and/or baked goods for customers, it is exceedingly difficult to plan ahead effectively. When looking at weekly orders of ingredients through this system, they can order roughly the same amount of ingredients every week, and then through a changing of the focus between what is needed right now and for events, they will be able to have a diverse bakery. Considering that nowadays people are asking for new or traditional baked goods, it is hard to find a balance between the two. Through having a different approach to what is readily available for the week, they can satisfy both of these from week to week, and month to month. Looking at things on a grand-scale, there are around 2,800 commercial bakeries, defined generically as small-bakeries, and there are around 6,000 retail bakeries, defined generically as large-bakeries[17]. It has been found that due to the factor of economies to scale, it puts a lot of pressure on these smaller bakeries. They have found that they need to do quite a few things: increase the amount of product produced which is extremely costly, increase the quality of product which can be costly, or provide specialties that a large-bakery will not be able to effectively produce. These further complicate the lives of small-bakeries, where they then have several hard choices to make, that could result in them going bankrupt. Additionally, these retail bakeries can further add pressure onto these small companies to either buy them out, or to pressure them with prices. Because they can afford to work at a loss for longer, they can compete with prices until the small bakery goes out of business should they want to. Therefore it is safe to assume that there will be a decline of the number of small-time bakeries in the near future. This system will allow for these small companies to be able to layout the next days schedule, or a couple days, and then they will be able to execute this plan. When this plan is executed, it will allow for this bakery to operate at a profit which in turn can allow them to scale up the purchasing of resources and tools to compete with these major companies. As shown in Figure 1.1.

1.1 Current State of the Art

Looking at the problem of closing bakeries, there are several tools that are available to the public. All of these would cost bakeries a considerable amount, and they do not provide the same functionality as this proposed software system. Most of these are just used as a database with a few functions just to make purchasing and tracking products easier. While these are important features, these do not ultimately reduce the amount of work that the baker does. There's an exceptional website, `http://www.cakeboss.com`[1], where it is easy for users. However, as mentioned earlier, it does not have the same logic.

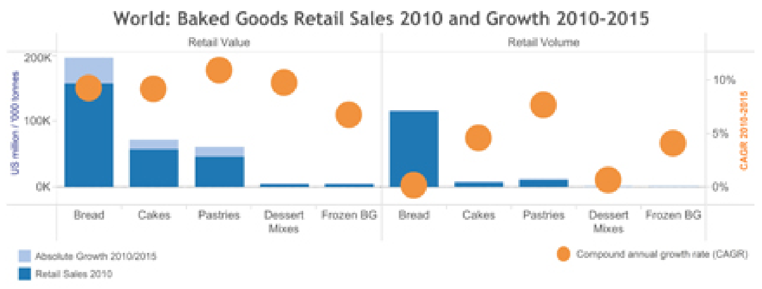


Figure 1.1: Graph of Bakeries Worldwide [9]

1.2 Goals of the Project

The goal of this project is to explore the use of CPSO revolving around minimizing the amount of wasted materials, the cost associated with a product and the price that it should have listed. Looking at the capabilities of the system, it will give the baker a list of recipes that can be made right now, how much it would cost them, and then let the baker know what they don't have the resources for.

1.3 Thesis Outline

Chapter 2 reviews a number of past approaches to the problem and summarizes their strengths and weaknesses. There are several studies that describe the exact problem of this paper, allowing for the reader to learn a lot. Chapter 3 outlines the method of approach used to establish the results in a comprehensive manner. Chapter 4 overviews the implementation and database schema. Chapter 5 contains the concluding remarks and future work.

Chapter 2

Related Work

2.1 History of PSO

PSO, standing for Particle Swarm Optimization, was proposed by R. Eberhart, and J. Kennedy[15]. How this is generally described is using the idea of how swarms act in nature such as bird flocking, schools of fish, etc. Looking for food sources in nature is more difficult than one may think. Therefore, when birds are looking for food, they will follow the lead of one or two birds, and assume that it will be leading them towards a reliable food source. The purpose of this algorithm is for looking for optimal solutions to problems that do not have an easy answer. What is considered an easy answer? Well, solving a simple problem with only one potential best solution is considered something easy to answer. However, when facing a problem that has many different solutions all of which have positives and negatives, it becomes hard for a system to accurately figure out the best answer. Additionally, sometimes the best answer cannot be found, as in there is no right answer. Hence the creation of PSO, using Particle Swarm Theory. The idea used is applying velocities to each of these particles. These velocities effect the particle in a certain way, and changes the distance away from the best particle. Thus, through a series of iterations and scaling the velocity, one can move these particles around to see where they converge. If the particle is in the general area of the "best" particle, then it is also considered to be a viable option. There have been several papers that have used this idea to solve problems revolving around MRCPSP, Multi-Mode Resource-Constrained Project Scheduling Problem. This is an NP-Hard problem, that computer scientists are trying to solve. This is just about what this proposed software system is about, and can help tremendously [10][13][6][3].

Looking at Figure 2.1 and Figure 2.2, this helps from a beginners perspective of the algorithm. What we can extrapolate from this is that the Particles are looking for the local minimum. This we can see by the particles beginning to group together, and then recognizing that a neighbor has a better location, so they try to group towards it. Furthermore, we can get closer to an optimal solution. While looking at the graph on the right we see a graph looking at the global best value. This is approached rather rapidly. This is the power of PSO, where they make many small calculations,

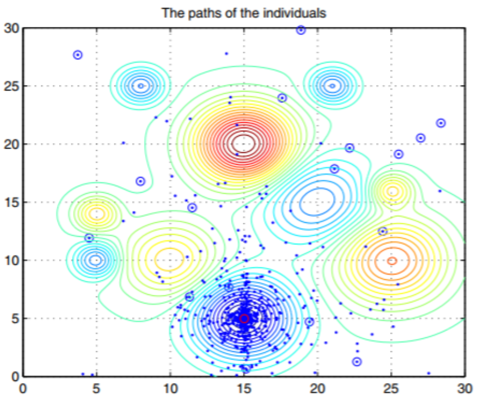


Figure 2.1: Paths of Particles [6]

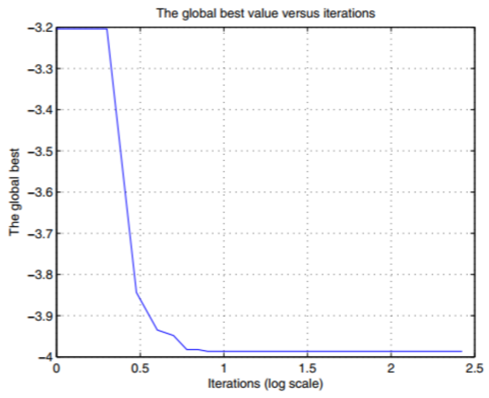


Figure 2.2: Global best value [6]

or changes. This in turn allows for quicker results. Because it is not slowed down by making large changes, it is a highly robust system.

2.2 Particle Swarm Optimization

Looking at the first Particle Swarm Optimization proposal, created by James Kennedy and Russel Eberhart, we can begin to have an understanding of PSO's. This was originally created to explore the possibility for replicating social behaviors. Although humans do not naturally replicate each other the way that schools of fish and birds will, humans have these changes with opinions and behavior [12]. The original equation for calculating the velocity is:

$$vx[][] = vx[][] + 2 \times rand() \times (pbestx[][] - presentx[][]) + 2 \times rand() \times (pbestx[][gbest] - presentx[][]) \tag{2.1}$$

This is the original velocity algorithm for changing the position of the particles. As mentioned in their own paper, while the results were promising, they believed that there may be an adjustment that could be made that they haven't come across yet. Upon further research, I have come across several works that will help with using the correct optimization solutions. Looking at[7], I have been able to construct an understanding for PSO. Within this reading they contained a very intuitive explanation of PSO which is more helpful than the original. This optimization revolves around attributing a particle to many iterations of modifying the location based off of the velocity function. How this helps out with this problem, is from the structure where it keeps track of all of the particles, which in this project refers to the recipes. It is then given a position, which is randomly given. Then it goes through the velocity iterations. I use the phrase velocity iterations to represent that there is a change of the position of the particle or recipe. It iterates through a list of the recipes and finds out if it satisfies the conditions associated with the specific velocity, and then moves the particle appropriately. In this way it then keeps on moving these particles to a small grouping of particles, which can then be returned to the user.

There has been a recent project using this PSO where they use it to efficiently come up with a solution to a problem with increasing the speed of looking through a smart grid [3]. In this project they explore what they call an RDT_PSO, standing for, informed discovery of resources in the grid system using particle swarm optimization. The purpose of this solution, is to search through a tree for a resource in the minimum amount of time and the minimum number of nodes visited[3]. It accomplishes this through storing only the local resources with the child nodes, in the roots. This makes it so the system does not need to go through as many trees and instead has to only look at one root. Additionally they further cut down on time through having an associated bitmap. Bitmaps can greatly reduce computational time, due to their simplicity. They then apply the PSO algorithm to figure out which resource is the best to use. Therefore accomplishing their task. Their data replicated the intention of lowering the amount of traffic and computational time considerably.

Additionally, through another source they have explored the use of Combinatorial PSO (CPSO), which then can be applied to problems with of the Multi-Mode Resource-Constrained Project Scheduling Problem, which is an NP problem[10]. Through this model, I can then apply some of the algorithms and discoveries that they have made to further cut down computational time, and increase the accuracy of the results. Here are the following equations associated with this section of the paper.

$$y_{ij}^t = \begin{cases} 1, & \text{if } x_{ij}^t = G_j^t \\ -1, & \text{if } x_{ij}^t = p_{ij}^t, \\ -1 \text{ or } 1, & \text{randomly if} (x_{ij}^t = G_j^t = p_{ij}^t), \\ 0, & \text{otherwise,} \end{cases} \tag{2.2}$$

Looking at Equation 2.2, it has conditional cases, which presents a lot of theoretical information about the particle represented by, y_{ij}^t. Here we find out what the associated value with it is, where we find out if the location, represented by, x_{ij}^t, is equal to the best position of the best particle in the swarm represented as G_i or if it is equal to the best position it has visited known as p_i [10]. It is worth noting that the v_{ij}^t is the inertia component that helps with using the memory of the direction of travel. This makes it so the particle will move more realistically as opposed to moving without regards to its past [4].

$$v_{ij}^t = \omega v_{ij}^{t-1} + c_1 r_1(-1 - y_{ij}^{t-1}) + c_2 r_2(1 - x_{ij}^{t-1}), \tag{2.3}$$

When looking at, Equation 2.3, we can notice that as mentioned earlier, this algorithm consists of changing positions of the particles based on certain factors. In this equation, c_1, and c_2 are acceleration constants. To be defined more abstractly it can be considered the cognitive weight[4]. Also, $r_1(t)$, and $r_2(t)$ $U(0,1)^k$ for all t [4]. This allows for the particle to be put into positions to see if it is an optimal solution. After getting attributed a value, the particle then moves towards a location. If that location is near the best location, it then evaluates it and saves it. Otherwise, it compares it to its personal best position, and if it is better it saves it or it does not. This has been modified from the original, where now it can actually move according to its previous position. Before it would move relative to the best particle position, and the best position of the best particle position. Therefore it allows for the system to start moving the particles because of their own distance as opposed to another distance.

$$\lambda_{ij}^t = y_{ij}^{t-1} + v_{ij}^t, \tag{2.4}$$

where this equation, 2.4, can represent the change in location.

$$y_{ij}^t = \begin{cases} 1, & \text{if } \lambda_{ij}^t > \alpha, \\ -1, & \text{if } \lambda_{ij}^t < -\alpha, \\ 0, & \text{otherwise,} \end{cases} \tag{2.5}$$

This is an important change to the particle y_{ij}^t. Through having this, the particle is then able to create a new solution of:

$$x_{ij}^t = \begin{cases} G_j^{t-1}, & \text{if } y_{ij}^t = 1, \\ p_{ij}^{t-1}, & \text{if } y_{ij}^t = -1, \\ \text{a random number}, & \text{otherwise.} \end{cases} \tag{2.6}$$

Equation 2.6, then allows for us to look at how the particle is affected based off of the effect of the velocity on the particle. This would allow for a a new assignment of the value of x_{ij}^t, and constricting it towards the optimal position. The α, represents a parameter that has been suggested to stay small, they used .3. This can then help us notice the change in the particle position, and whether or not it is significant.

The generalization of the Particle Swarm Optimization, is very important when beginning to learn about this. And this paragraph will help with the theoretical application of the PSO. Particle convergence is very important for this idea, and while it has been mentioned it has not been explained. I will use the explanation from [4]:

> Let the dimension of the search space be k. Then, particle i is said to converge if there exists a $\mathbf{p} \in I\mathbb{R}^k$ such that for every $\epsilon > 0$ there exists a $\hat{t}(\epsilon)$ such that if $t > \hat{t}(\epsilon)$, then $||\mathbf{x}_i(t) - \mathbf{p}|| \leq \epsilon$, where $|| \bullet ||$ is an arbitrary norm on $\mathbb{R}^k$.

This is very important for the entire purpose of using this algorithm. Here it describes the behavior of the particle and that it will approach being at rest. It is also described that, it is not important to know the point of convergence, but that it is important to know that the particles will converge. Although many aspects of PSO have been observed, one of its major shortcomings is its need for the stagnation assumption. This assumption is that the personal best, and neighborhoods best are assumed to be non-changing. Why people have this assumption is because it makes the mathematical equations actually work. Additionally modeling theoretical information, in a very literal library is nigh impossible. However it does not mean that we have to despair and shy away from this. This is because there are several ways to get around it. Through using the weak chaotic assumption, it allows for there to be a change in the y_{ij}^t and $\hat{y}_{ij}^t$. This is important because then the particle can actually be able to approach an actual PSO[4]. Further this does not mean that the positions will necessarily change drastically.

Additionally, this paper further brought forwards some other important factors that I did not know about beforehand. There is an important distinction between two applications of the PSO, where the algorithm used is where the stochastic component is assumed to be fixed, and the other that the assumption is dropped. Although both of these have potential for giving important feedback, they both have positives and associated negatives. Through using stochastic approach, it is considerably easier to implement and construct theoretically. This is because you can assume what the best particle positions and then you can further save them and use them to lead the other particles to a PSO. However, that being said, this has a major weakness. Because you are assuming the best position, which may be false, then it can completely ruin a set of data. Furthermore, this is where dropping the assumption is more favorable. Through being able to drop the assumption, it allows to more accurately look for more potential solutions. However, the positives with the negatives, this system has not been discovered yet. The closest we can get to this is through the weak chaotic assumption, that allows for the best locations to be alterable. Another section that is worth mentioning is the section where they are able to determine that for practical applications, they can formulate a series to allow for the particles to converge more effectively, so long as $\max\{|\lambda_1|, |\lambda_2|\} < 1$, with λ_1 and λ_2:

$$\lambda_1 = \frac{1}{2}(-\sqrt{(\theta_1 + \theta_2 - w - 1)^2 - 4w} - \theta_1 - \theta_2 + w + 1), \tag{2.7}$$

$$\lambda_2 = \frac{1}{2}(\sqrt{(\theta_1 + \theta_2 - w - 1)^2 - 4w} - \theta_1 - \theta_2 + w + 1), \tag{2.8}$$

this becomes very useful from our perspective for us to know what's going on theoretically in the program.

$$|w| < 1, \tag{2.9}$$

$$0 < \theta_1 + \theta_2 < 4, \tag{2.10}$$

$$w > \frac{\theta_1 + \theta_2}{2} - 1. \tag{2.11}$$

Through these parameters, we can further extrapolate a lot of information. Although

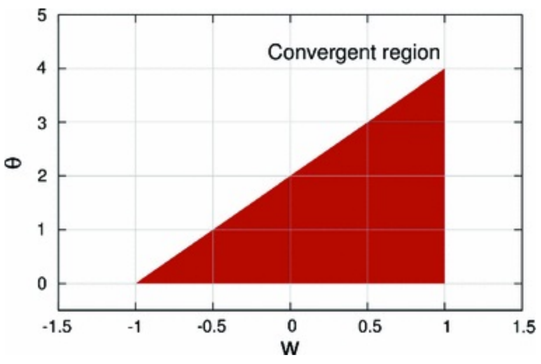


Figure 2.3: Parameter region in which each particle converges to a point [4]

it is important to note that this is not the λ, from Equation 2.5. With these conditions met, it allows for the particles to have a higher probability of converging to the same spot. While it is not necessarily optimal in this instance to find one best particle, by having the neighborhood grouping around a smaller area greatly helps. This is shown in Figure 2.3.

Looking at another paper revolving around the management of resources in a manufacturing company they say a very interesting statement, "their management of allocation is usually inefficient and ineffective"[13]. This is interesting to me because intuitively I believed that these would be things that a manufacturing company would have mastery of. But they have found that in most cases, the allocation of resources is not optimal. However they also branch off and use Complex Network Theory in junction with this. This allows them to associate the projects, and the required resources that they would like to use. Additionally, where Complex Network Theory is successful, with having many nodes, the PSO is weak. This makes it a good pairing, and they were able to find an increase in general optimization by 10%.

When searching for inspiration for how to learn other methods how to structure the system, I found that [3], provided a comprehensible system. This system is for exploring the use of PSO for the intention of solving a smart grid. This problem is not directly related to a bakery manager, however it provides this innovative idea of containing information about the childs of parent nodes, within the parent. This would allow for the system to then be able to decrease the amount of traffic created by searching through graphs. Additionally, before reading this research paper I was considering using a coding system to further cut down the searching time, but now I am convinced it is something to follow through on. Currently I am thinking of using the bitmapping structure, which can then be used to represent a lot of data. However the system does not have to look through as much information.

2.2.1 Recent Research in Management of Multiple-Resources

Considering the scarcity of resources in the world, this has become a very important problem to find a solution to. Therefore researchers have had several different ways to attempt to find the optimal solution to managing these resources to have a favorable outcome. Most of these are on specific non-renewable resources, such as oil, gas, coal, etc. These systems can then take advantage of the fact that there is very limited ways for us to increase the supply of these power sources, and therefore can make an important assumption. The input of these resources are stationary, and are depleting. This then allows for them to use simple economic principles, and advanced mathematics to judge what the associated price is. One of the potential methods to use, is using the Larange Dual Decomposition method [11]. This however is more intended for the use on a smart grid problem, and will not be used because this cannot come up with a solution. Speaking about smart grid problems, it has been explored through using PSO. Using principles about the advantages of PSO, they were able to create a system that could look through an entire smart grid and suggest the best options. This was accomplished through evaluating the resources needed and the associated cost[3].

2.3 Relational Databases

To address this problem with management of resources and time, we can create optimization software and use relational databases. In the work of, *Dependencies in relational databases*, mentioned in Thalheim[16], it has discussed how to construct Relational Databases with a greater reliance. This in turn allows for the system to be able to run considerably faster, despite it being larger. Considering that this system may be run on a system with lower computational power, taking any shortcut could help the speed immensely. This can be of great assistance to decreasing the time complexity, where through creating several join tables, intersection tables, and union tables. Looking at the logical expression for these, they can take the appearance of Sets in mathematical notation. For example, let R denote the set of recipes. Within this set of recipes, it contains several subsets. Let $S \subseteq R$, representing the best season for this to be served.

$$S = \{Spring, Summer, Fall, Winter\} \tag{2.12}$$

Through knowing the current date, denoted as D, containing . If it falls in the correct season the recipe will then go to the next query, in other words,

$$S \cap D = \{x | x \in Dandx \in R\}. \tag{2.13}$$

Should the recipe contain either `all` or `none`, in this situation it means that there is no specified season and therefore can be made in any season. Although this is a simple example, it shows that there are several ways to bypass a lot of unnecessary parsing through rows and columns of data.

2.4 Tools

Looking at the platform that I am using, django version 2.0[2], which is ideal for this situation. The features that come with the system are very intuitive for users, allowing them to not have problems with using it. Looking at the structure of the website 3.2, it is self explanatory and easy for users to interact with. Through a series of clicks and simple changes they can interact with the database. This is highly helpful considering that if we were to use a different method, I would have to create a GUI for them to interact with or another python program. While this is still a possibility, this is beyond the scope of the course. Positively django also uses sqlite3. This is a very useful database style because it is highly interactive. Python has a plugin that in future implementations can further explore automated management systems for updating the database in the background. Currently how the system is written is to update after the user interacts with it. Another helpful feature is that users cannot input the wrong data into a data field. For example one cannot put a ”one” into an integer field. This is a source of error in database systems where they have to account for inputting bad data. Additionally it has features involving users, where specific users can only have access to certain actions with the website. This increases security in two aspects. The first is that there is a security from outsiders, or people within the company, trying to change data that could endanger the business. Through giving people access to certain databases, it restricts these error prone areas. Additionally, the second aspect of security, is that it decreases the chances of one of the users accidentally changing important data.

Although this has been mentioned before, I will be programming this in python. The purpose of this is that it is what I believe to be one of the most versatile languages. It has many packages that can be used for varying situations. For example, considering that the database is in sqlite3, it is highly conducive to use its capabilities such that it can interact with the database of django. Through this capability it then just needs to use some of its basic functions to run effectively. Through knowing where the data is that the user is putting in, it takes some very simple string manipulation to extrapolate the needed amount of information. This method is `stringName.split("splitAtVariable")`. The method takes a string, and then creates an array of objects containing the objects around the specified `splitAtVariable`. This is very useful for not wasting space in the structure of the database. Furthermore, through creating a general method, this can further be used for parsing through the instruction sections.

With automizing the behaviors of a software and database system, it is important to note how they should be presented to the users, and the managers of the database.

This idea is represented in the works of, *Springer Handbook of Automation, pages 1539-1548*, Springer[14]. Additionally, they talk a great deal about abstracting what the user can interact with for the database. The idea behind this, is through limiting the times that the database, or values of the database, are changed it greatly reduces the errors that could be made. This is important in terms of security for many companies, and in addition it could be an accidental switch from a ”100 pounds of bread flour” to ”1000 pounds of bread flour.” Therefore through making it so the workers cannot even change the values, only see what they need to do for the day, it greatly reduces the chance of error. Furthermore, through automating the system it allows for bakers to iterate through several different plans and then make the best choice. It can be advised to do many things, and all of them may have the same projected revenue generated from the public. However, there are factors that cannot be implemented that a human would find intuitive.

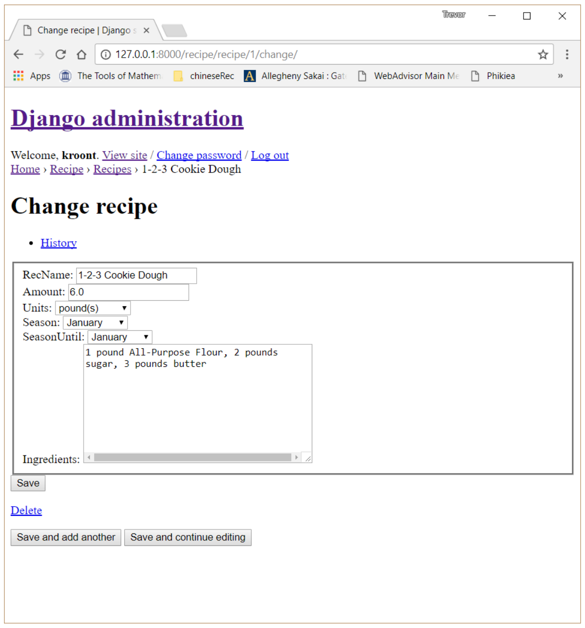
Chapter 3

Method of Approach

3.1 Environment

This system consists of several working parts. Through using the django system, it allows for there to be an easy way to interact with the database for the user. In addition it also supports multiple hardware systems. Making it easier for users to access what they need to do for the day. Additionally it creates a website that is easy to navigate through. This can be difficult with database systems, and therefore through having an intuitive system, it makes it that much easier for the baker to understand. Further, it has several different fields as represented in the database Schema **??**, all represented in subsections of these subjects. There is one more additional part to the system that is the brain of the system. The majority of the code is in `main-code.py`, which can then be run from one of the subsections. This will parse through all of the recipes, and the available ingredients, and allow for the manager to look through and choose them. Thus it will then send out the needed ingredients, and then the schedule for the week to the workers.

Through the application of the equations in Section **??**, it allows us to solve many problems for a baker without them having to remember anything about their inventory. If a recipe requires them to have a certain tool, and they do not have the tool, then it will not recommend it. When a baker has been around for several years, they acquire numerous tools and obtain many recipes, that takes up a lot of memory. Through this software system, they can know a day in advance what to do. When in charge of a bakery this can be very helpful considering that having to come up with these ideas takes up a lot of time, and time is money. This will present a spot of error, where one may not want any of these recipes that have been mentioned. Therefore there will be a feature with a ranking system for the suggested recipes, approaching a, "would not recommend" list. The reason to have these listed is for the simplicity of the user, where they can then know that if they were looking at creating a specific recipe, what the associated cost would be of making it and the estimated pricing. Looking at how the users of the program will insert recipes, or change recipes, through a series of clicks they can navigate through and find the



Django administration

Welcome, **kroont**. View site / Change password / Log out
Home › Recipe › Recipes › 1-2-3 Cookie Dough

Change recipe

- History

RecName: 1-2-3 Cookie Dough
Amount: 6.0
Units: pound(s)
Season: January
SeasonUntil: January
Ingredients: 1 pound All-Purpose Flour, 2 pounds sugar, 3 pounds butter

Figure 3.1: Example of Recipe

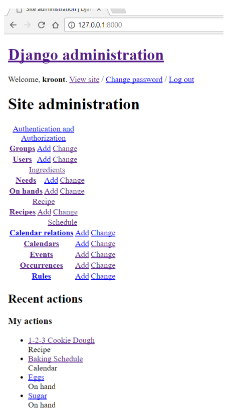
recipe as shown in Figure 3.1.

One of the tables that has not been mentioned yet is the log table. The log table consists of several subtables called, Customer History, Purchase History Resources Used, and Sales. All of these contain the data that will help the optimization algorithm. Looking at Customer History's, we can find out on average what people will buy and the amount. Therefore the software will learn what's trending, and hold those recipes to a higher standard. Additionally, if someone goes there everyday, and systematically gets the same thing every time, you can hold that knowledge and use it to your advantage. Another field is their age range. Dr. Mohan recommended keeping track of the age of a customer, where you can track trends. For example, if people that are 60 years or older, may like a certain pumpernickel bread. Looking at this data, when someone fits the description you can recommend it safely. Also knowing the time of day that it was purchased can help with planning a week. Looking at the Purchasing History, it allows for the system to know what they have on hand, as well as the cost to get those ingredients. Another table is Resources Used, where it is for the bakery workers to update the inventory. This information can be used for planning events and recipes. Although a recipe may ask for 1 pound of flour, the worker may need to use an additional pound of flour, which should go into account. Because some people do not want to be tracked in a system, which is their right, there is also a Sales table. This keeps track of all of the products sold, and not sold. The database can return the needed knowledge for the baker. Looking at the Database Schema found in Chapter4, one can notice how intuitive of a system this is for updating a bakery. Looking at the method of approach, it has several different views depending on the user. For example, one bakery may decide that they only want to use this to store recipes and have it so their workers only see what they need to bake. Therefore half of the system wouldn't be used. However, to those that want to use this system to its fullest capabilities, the process would be very different. The day that they decide to plan out the week, month, year, etc., they can go into great detail about how many of a recipe they would like to make, the needed resources for that to happen, and when they would need to start that recipe. At first glance this seems easy, however, when you have a recipe that makes 10 pounds of dough, and then you need to make 400 individual pastries for an event, scaling the amount of dough needed and the ingredients needed, takes a while for bakers to calculate. Then they have to check to make sure that their math was right. Instead, this would allow them to quickly find out how many pounds of flour, pounds of sugar, etc., they would need for that week. One additional feature that could help them a considerable amount, would be if they already have several recipes going, and have quite a few resources left over, which they can then search to see if they can do another recipe. Throughout many fields, the unexpected happens, where someone writes down an extra zero, or forgets to write a one, so there may be a surplus of resources at some point. The code would already be written for a generalized case, but through allowing a specific search, it can allow them to use these resources and increase their revenue. From the view of the workers for this baker, it makes it even simpler to follow their instructions for the day. Through only needing to know the bare minimum, and exactly when to start an action, the bakery can then run like clock work. This colloquial phrase is an ideal case for a bakery where no time is wasted. Timing is everything in a bakers world, where if certain actions happen too late or too early, it can ruin the product. This also allows for there to be no room for blaming others for mistakes, and not needing to worry about what other workers are doing.

Additionally, to further increase the speed, we can do bitmapping. Through abstracting the information, and giving it an associated bit number, it makes it easier for the system. For example, if going through the recipe and it notices that you do not have the resources, instead of storing the information as some text, it can keep it as a 1 or a 0. This makes it easier in the future when checking the values that is ideal for the situation. Although I thought of this several weeks before writing it down, I was not entirely convinced it would be the best thing to do. However, after looking through the papers, one of the systems used this method of bitmapping, which re-convinced me to do this plan.

3.1.1 Reasoning

Throughout the introduction of this paper I have mentioned the "how" and the "what for," but I have not mentioned the "why" for using certain systems and methodologies. The reason for using a Combinational Particle Swarm Optimization, CPSO, is that it has been found to scale better when trying to solve the MSCRP styled problems[10]. Through using the proper tools to solve this problem it will allow for better answers



127.0.0.1:8000

Django administration

Welcome, **kroont**. View site / Change password / Log out

Site administration

Authentication and Authorization
Groups Add Change
Users Add Change
Ingredients
Needs Add Change
On hands Add Change
Recipe
Recipes Add Change
Schedule
Calendar relations Add Change
Calendars Add Change
Events Add Change
Occurrences Add Change
Rules Add Change

Recent actions

My actions

- 1-2-3 Cookie Dough
  Recipe
- Baking Schedule
  Calendar
- Eggs
  On hand
- Sugar
  On hand

Figure 3.2: Structure of the Baking Website

for a baker, and allow them to effectively know what the plan should be for a day. Additionally, through using django, it allows for the users to interact with a database very easily. Databases can be very difficult to interact with by nature, and through creating a system that from the beginning can make it so it is easy for them to use is a very favorable option. With, Figure 3.2, we can see the general structure of this website. Although it is not aesthetically pleasing right now, it shows the simplicity of how to use django. Where one simply has to click on where they want to go, and then they can change the information as wanted. Included in django, is the `users` section. This can be very helpful for the bakery for managing who can interact with what data. For example, only the person in charge of ordering the ingredients weekly can input the data about what they got. This makes it so no one can accidentally change data or purposefully change data. Additionally, django can be used among many hardware systems. That is to say that it is widely available on mobile devices, tablets, laptops, and computers. This can make it very easy for the baker to do work anywhere, and at anytime. Included in django, is that it limits what you can insert into the database in the sense that it will only let you put it in if its of the correct form. For example, you cannot insert, ”one,” into an integer field. This is great from the programmers perspective because it limits the amount of error you have to account for from the user. From our viewpoint as programmers it is also favorable for several reasons. First, because of its structure it is easy to modify for the users to help them get what they need out of the tool. Additionally, from several aspects of python, this coding becomes extremely versatile. Python can interact with the sqlite3 database that django uses, and is highly useful for string manipulation. These two combined allows for not needing to jump through several different programs, and can instead be written in one `main-code.py`. Through using the system called django-scheduler, it allows for the scheduling of events in the calendar year. This is extremely useful for bakeries. Knowing what is coming up allows for them to plan accordingly and buy the correct amount of ingredients for the week. Additionally, it has a feature where the event can be set up to repeat which means that they will not need to re-plan recurring events.

3.2 Experiments

When looking at the experiments with this system, there are several ways to assess the system. Although the recipes that I have populated the database with are not actual recipes, and would not recommend making them, it allows for there to be important testing. Through creating these problems, it allows for the handling of real-life situations. For example, when looking at what to do with 15 pounds of flour and 3 pounds of sugar, it can look through the recipes and then you'll know what you can make. Additionally, when looking at the functionality of the system, we can make it do some very robust things. Because of the construction of the system, one could just as easy make this a chemistry system, or a metal industry, or anything that you need to follow a "recipe" for.

3.3 Threats to Validity

Although this system has many strengths, there are several threats to the validity of my software system. For some reason, if there are double-digits in the amount of Sugar the system skips over it. I have not been able to figure why this is, and have tried using hundreds of debugging statements to figure it out, but to no avail. Although I have mentioned that there are applications to other fields, I have not asked a member of that field to test the system out. Looking at the system, there are certainly better ways to construct it, which will be discussed in the conclusion. When creating the code, I approached it with the Monte Carlo attitude, where even if it will take a long-time to complete and will be inefficient, it will come up with the answer. However, I believe that the from a logical perspective this is a valid solution to a complex problem. One of the negative aspects of these tools, is that django needs to be connected to the internet. However, considering that everyone in our society is connected to the internet at any given point in time, I do not think this will be an issue. Looking at the size of the database. It can be overwhelming. There will be over one-hundred recipes in the average bakers recipe book, and some will approach above one-thousand. This is daunting to think about from a computer science perspective, where there is so much data in one location. Positively though, sqlite3 can hold significantly more than 1TB, which is a lot of data. Through some of the relational databases, while it does increase the size of the database, it greatly increases the search speed. This will make it seem smaller than it actually is. Furthermore, python itself is not a fast language. From its construction, it focuses on having a language that is easier to program in. This makes it so there is more to process while going to the computers natural language, which increases computational time. However, from the idea behind CPSO, it is only making small changes, hundreds of times. Therefore the computational time really will not come into effect. Additionally, as mentioned earlier, python has a very powerful string manipulation kit, which can greatly help when parsing through a database and making changes.

3.3.1 Alternative Methods

Another method is through using genetic algorithms. Because of the nature of PSO's, they contain aspects of genetic algorithms where they are changing the information to find the most optimal solution. However after looking into potential genetic algorithms, we noticed that this has too much room for error, and therefore will take longer to come to the same conclusion. This field also does not have the same ability to abstract the same information. Furthermore, mentioned in a couple of readings, they specifically claimed that they had considered using some form of genetic algorithms, but could not find a substantially good method. Additionally we have looked into Simplex Algorithms. These seem like they could hold a potential solution to this problem, however there are more sources revolving around PSO, and how they can be used to solve this MSCRP.

Chapter 4

Implementation

4.1 Database Schema

```
CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "app" varchar(255) NOT NULL, "name" varchar(255)
    NOT NULL, "applied" datetime NOT NULL);
CREATE TABLE "auth_group" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "name" varchar(80) NOT NULL UNIQUE);
CREATE TABLE "auth_group_permissions" ("id" integer NOT NULL PRIMARY
     KEY AUTOINCREMENT, "group_id" integer NOT NULL REFERENCES "
    auth_group" ("id"), "permission_id" integer NOT NULL REFERENCES "
    auth_permission" ("id"));
CREATE TABLE "auth_user_groups" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user"
    ("id"), "group_id" integer NOT NULL REFERENCES "auth_group" ("id"
    ));
CREATE TABLE "auth_user_user_permissions" ("id" integer NOT NULL
    PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES
    "auth_user" ("id"), "permission_id" integer NOT NULL REFERENCES "
    auth_permission" ("id"));
CREATE UNIQUE INDEX "
    auth_group_permissions_group_id_permission_id_0cd325b0_uniq" ON "
    auth_group_permissions" ("group_id", "permission_id");
CREATE INDEX "auth_group_permissions_group_id_b120cbf9" ON "
    auth_group_permissions" ("group_id");
CREATE INDEX "auth_group_permissions_permission_id_84c5c92e" ON "
    auth_group_permissions" ("permission_id");
CREATE UNIQUE INDEX "auth_user_groups_user_id_group_id_94350c0c_uniq
    " ON "auth_user_groups" ("user_id", "group_id");
CREATE INDEX "auth_user_groups_user_id_6a12ed8b" ON "
    auth_user_groups" ("user_id");
CREATE INDEX "auth_user_groups_group_id_97559544" ON "
    auth_user_groups" ("group_id");
CREATE UNIQUE INDEX "
    auth_user_user_permissions_user_id_permission_id_14a6b632_uniq"
    ON "auth_user_user_permissions" ("user_id", "permission_id");
```

```
CREATE INDEX "auth_user_user_permissions_user_id_a95ead1b" ON "
    auth_user_user_permissions" ("user_id");
CREATE INDEX "auth_user_user_permissions_permission_id_1fbb5f2c" ON
    "auth_user_user_permissions" ("permission_id");
CREATE TABLE "django_admin_log" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "object_id" text NULL, "object_repr" varchar(200)
    NOT NULL, "action_flag" smallint unsigned NOT NULL, "
    change_message" text NOT NULL, "content_type_id" integer NULL
    REFERENCES "django_content_type" ("id"), "user_id" integer NOT
    NULL REFERENCES "auth_user" ("id"), "action_time" datetime NOT
    NULL);
CREATE INDEX "django_admin_log_content_type_id_c4bce8eb" ON "
    django_admin_log" ("content_type_id");
CREATE INDEX "django_admin_log_user_id_c564eba6" ON "
    django_admin_log" ("user_id");
CREATE TABLE "django_content_type" ("id" integer NOT NULL PRIMARY
    KEY AUTOINCREMENT, "app_label" varchar(100) NOT NULL, "model"
    varchar(100) NOT NULL);
CREATE UNIQUE INDEX "
    django_content_type_app_label_model_76bd3d3b_uniq" ON "
    django_content_type" ("app_label", "model");
CREATE TABLE "auth_permission" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "content_type_id" integer NOT NULL REFERENCES "
    django_content_type" ("id"), "codename" varchar(100) NOT NULL, "
    name" varchar(255) NOT NULL);
CREATE UNIQUE INDEX "
    auth_permission_content_type_id_codename_01ab375a_uniq" ON "
    auth_permission" ("content_type_id", "codename");
CREATE INDEX "auth_permission_content_type_id_2f476e4b" ON "
    auth_permission" ("content_type_id");
CREATE TABLE "auth_user" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "password" varchar(128) NOT NULL, "last_login"
    datetime NULL, "is_superuser" bool NOT NULL, "username" varchar
    (150) NOT NULL UNIQUE, "first_name" varchar(30) NOT NULL, "email"
     varchar(254) NOT NULL, "is_staff" bool NOT NULL, "is_active"
    bool NOT NULL, "date_joined" datetime NOT NULL, "last_name"
    varchar(150) NOT NULL);
CREATE TABLE "django_session" ("session_key" varchar(40) NOT NULL
    PRIMARY KEY, "session_data" text NOT NULL, "expire_date" datetime
     NOT NULL);
CREATE INDEX "django_session_expire_date_a5c62663" ON "
    django_session" ("expire_date");
CREATE TABLE "ingredients_need" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "ingredient" varchar(100) NOT NULL, "weight" real
    NOT NULL, "units" varchar(20) NOT NULL);
CREATE TABLE "log_purchasehistory" ("id" integer NOT NULL PRIMARY
    KEY AUTOINCREMENT, "ingredient" varchar(100) NOT NULL, "amount"
    real NOT NULL, "units" varchar(20) NOT NULL, "cost" real NOT NULL
    , "day" integer NOT NULL);
CREATE TABLE "log_sales" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "bakedGood" varchar(32) NOT NULL, "amtSold"
    integer NOT NULL, "amtMade" integer NOT NULL, "day" integer NOT
    NULL);
CREATE TABLE "log_resourcesused" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "ing" varchar(32) NOT NULL, "amt" real NOT NULL, "
    units" varchar(20) NOT NULL, "day" integer NOT NULL);
CREATE TABLE "log_customerhistory" ("id" integer NOT NULL PRIMARY
    KEY AUTOINCREMENT, "name" varchar(32) NOT NULL, "AgeRange"
    varchar(20) NOT NULL, "day" integer NOT NULL, "bakedGood" text
    NOT NULL);
CREATE TABLE "recipe_recipe" ("id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT, "amount" real NOT NULL, "units" varchar(20) NOT
    NULL, "recName" varchar(100) NOT NULL, "ingredients" text NOT
    NULL, "season" varchar(20) NOT NULL, "seasonUntil" varchar(20)
    NOT NULL, "timeToMake" real NOT NULL, "timeToMakeUnits" varchar
    (10) NOT NULL);
```

Listing 4.1: Database Schema

4.2 Code

```
#
    ---------------------------------------------------------------------------------

#
#   Nathan A. Rooy
#   Simple Particle Swarm Optimization (PSO) with Python
#   July, 2016
#
#
    ---------------------------------------------------------------------------------


#--- IMPORT DEPENDENCIES
    -------------------------------------------------------+

from __future__ import division
import sqlite3
import math
import random

#--- COST FUNCTION
    ------------------------------------------------------------+

# function we are attempting to optimize (minimize)
def func1(x):
    total=0
    for i in range(len(x)):
        total+=x[i]**2
    return total
```

```
#--- MAIN
    ----------------------------------------------------------------------+


class Particle:
    def __init__(self,x0):
        self.position_i=[]          # particle position
        self.velocity_i=[]          # particle velocity
        self.pos_best_i=[]          # best position individual
        self.err_best_i=-1          # best error individual
        self.err_i=-1               # error individual

        for i in range(0,num_dimensions):
            self.velocity_i.append(random.uniform(-1,1))
            self.position_i.append(x0[i])

    # evaluate current fitness
    def evaluate(self,costFunc):
        self.err_i=costFunc(self.position_i)

        # check to see if the current position is an individual best
        if self.err_i < self.err_best_i or self.err_best_i==-1:
            self.pos_best_i=self.position_i
            self.err_best_i=self.err_i

    # update new particle velocity
    def update_velocity(self,pos_best_g):
        w=0.5       # constant inertia weight (how much to weigh the
            previous velocity)
        c1=1        # cognative constant
        c2=2        # social constant

        for i in range(0,num_dimensions):
            r1=random.random()
            r2=random.random()

            vel_cognitive=c1*r1*(self.pos_best_i[i]-self.position_i[
               i])
            vel_social=c2*r2*(pos_best_g[i]-self.position_i[i])
            self.velocity_i[i]=w*self.velocity_i[i]+vel_cognitive+
               vel_social

    # update the particle position based off new velocity updates
    def update_position(self,bounds):
        for i in range(0,num_dimensions):
            self.position_i[i]=self.position_i[i]+self.velocity_i[i]

            # adjust maximum position if necessary
            if self.position_i[i]>bounds[i][1]:
                self.position_i[i]=bounds[i][1]

            # adjust minimum position if neseccary
            if self.position_i[i] < bounds[i][0]:
                self.position_i[i]=bounds[i][0]

class PSO():
    def __init__(self,costFunc,x0,bounds,num_particles,maxiter):
        global num_dimensions

        num_dimensions=len(x0)
        err_best_g=-1                   # best error for group
        pos_best_g=[]                   # best position for group

        # establish the swarm
        swarm=[]
        for i in range(0,num_particles):
            swarm.append(Particle(x0))

        # begin optimization loop
        i=0
        while i < maxiter:
            #print i,err_best_g
            # cycle through particles in swarm and evaluate fitness
            for j in range(0,num_particles):
                swarm[j].evaluate(costFunc)

                # determine if current particle is the best (
                    globally)
                if swarm[j].err_i < err_best_g or err_best_g == -1:
                    pos_best_g=list(swarm[j].position_i)
                    err_best_g=float(swarm[j].err_i)

            # cycle through swarm and update velocities and position
            for j in range(0,num_particles):
                swarm[j].update_velocity(pos_best_g)
                swarm[j].update_position(bounds)
            i+=1

        # print final results
        print('FIAL:')
        print(pos_best_g)
        print(err_best_g)
#--- END
    ----------------------------------------------------------------------+

# use the unique id to make sure that items don't repeat



#--- RUN
    ----------------------------------------------------------------------+

# Trevor J. Kroon
# Allegheny College
# Department of Computer Science
# April, 2018
# Honor Code:
# The work I am submitting below is mine, unless otherwise cited.
#
   -----------------------------------------------------------------------------

class main():
    # @param
    # x -> desired amount
    # y -> original amount
    # @return
    # returns the factor to multiply the recipe by
    def fcf(self, x, y):
        self.des = x
        self.org = y
        self.temp = x / y
        return (self.temp)

    # @param
    # x -> lst[][]
    # y -> ingredient looking for
    # if the ingredient can't be found in the lstOnHand, it checks
       to see if it is another
    def inRec(self, x, y, size):
        self.tempX = x
        self.tempY = y
        self.c = 0
        for v in range(size):
            if self.tempY == self.tempX[v][0] + " ":
                return(True)
            self.c += 1
        return(False)

    def makeList(self, x, name, w, h):
        self.lst = [[None for x in range(w)]for y in range(h)]
        self.c = 0
        for m in range(len(name)):
            self.lst[m][0] = name[m]
        for line_list in x:
            self.temp = " ".join(line_list)
            self.temp2 = self.temp.split(",")
            for p in range(len(self.temp2)):
                if self.temp2[p] != None:
                    self.temp3 = self.temp2[p].split(" ")
                    if len(self.temp3) > 2:
                        self.temp4 = ""
                        self.temp5 = ""
                        self.temp5 = list(filter(None,self.temp3))
                        for n in range(len(self.temp5)):
                            self.temp4 += self.temp5[n] + " "
                        self.lst[self.c][p+1] = self.temp4
            self.c += 1
        self.temp5 = list(filter(None,self.lst))
        self.temp6 = [[None for x in range(w)]for y in range(len(
            self.temp5))]
        for v in range(len(self.temp5)):
            self.temp6[v] = list(filter(None,self.temp5[v]))
        return(self.lst)

    def makeList1(self, x, w):
        self.lstc = [None for x in range(w)]
        self.c = 0
        for line_list in x:
            self.temp = " ".join(line_list)
            self.temp2 = self.temp.split(",")
            for p in range(len(self.temp2)):
                if len(self.temp3) > 2:
                    self.temp4 = ""
                    self.temp5 = ""
                    self.temp5 = list(filter(None,self.temp3))
                    for n in range(len(self.temp5)):
                        self.temp4 += self.temp5[n] + " "
                    self.lst[self.c] = self.temp4
            self.c += 1
        return(self.lstc)
    def __init__(self):
        self.con = sqlite3.connect("db.sqlite3")
        self.query = self.con.cursor()

        self.w = 13 # find the size of it later, and modify
        self.h = 20 # find the size of it later, and modify

        # skeleton for parsing through names
        self.tableName = "recipe_recipe"
        self.rowName = "recName"
        self.query.execute("select {rn} from {tn};".format(rn = self
            .rowName, tn = self.tableName))
        self.row = self.query.fetchall()
        self.numOfRec = len(self.row)
        self.lst =  [[None for x in range(self.w)]for y in range(
            self.numOfRec)]
        self.lstRec = [None for x in range(self.numOfRec)]
        self.c = 0
        for line_list in self.row:
            temp = " ".join(line_list)
            self.lstRec[self.c]= temp
            self.c += 1

        self.rowName = 'ingredients'
        self.query.execute("select {rn} from {tn};".format(rn = self
            .rowName, tn = self.tableName))
        self.row = self.query.fetchall()
```

```
self.lst = self.makeList(self.row, self.lstRec, self.w, self
   .numOfRec)
self.c = 0

self.tableName = "log_customerhistory"
self.rowName = "bakedGood"
self.rowName2 = "day"
self.query.execute("select {rn} from {tn} where {rn2} = 1;".
   format(rn = self.rowName, tn = self.tableName, rn2 = self
   .rowName2))
self.row = self.query.fetchall()
self.day1 = [None for x in range(len(self.row))]
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.day1[self.c] = temp
    self.c += 1


self.query.execute("select {rn} from {tn} where {rn2} = 2;".
   format(rn = self.rowName, tn = self.tableName, rn2 = self
   .rowName2))
self.row = self.query.fetchall()
self.day2 = [None for x in range(len(self.row))]
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.day2[self.c] = temp
    self.c += 1

self.query.execute("select {rn} from {tn} where {rn2} = 3;".
   format(rn = self.rowName, tn = self.tableName, rn2 = self
   .rowName2))
self.row = self.query.fetchall()
self.day3 = [None for x in range(len(self.row))]
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.day3[self.c] = temp
    self.c += 1

self.query.execute("select {rn} from {tn} where {rn2} = 4;".
   format(rn = self.rowName, tn = self.tableName, rn2 = self
   .rowName2))
self.row = self.query.fetchall()
self.day4 = [None for x in range(len(self.row))]
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.day4[self.c] = temp
    self.c += 1
self.query.execute("select {rn} from {tn} where {rn2} = 5;".
   format(rn = self.rowName, tn = self.tableName, rn2 = self
   .rowName2))
self.row = self.query.fetchall()
self.day5 = [None for x in range(len(self.row))]
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.day5[self.c] = temp
    self.c += 1

self.query.execute("select {rn} from {tn} where {rn2} = 6;".
   format(rn = self.rowName, tn = self.tableName, rn2 = self
   .rowName2))
self.row = self.query.fetchall()
self.day6 = [None for x in range(len(self.row))]
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.day6[self.c] = temp
    self.c += 1

self.query.execute("select {rn} from {tn} where {rn2} = 7;".
   format(rn = self.rowName, tn = self.tableName, rn2 = self
   .rowName2))
self.row = self.query.fetchall()
self.day7 = [None for x in range(len(self.row))]
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.day7[self.c] = temp
    self.c += 1

self.tableName = "log_purchasehistory"
self.rowName = "ingredient"
self.query.execute("select {rn} from {tn};".format(rn = self
   .rowName, tn = self.tableName))
self.row = self.query.fetchall()
self.numOfIng = len(self.row)
self.c = 0
OnHand = [[None for x in range(3)]for y in range(self.
   numOfIng)]
self.lstOnHand = [[None for x in range(3)]for y in range(
   self.numOfIng)]
for line_list in self.row:
    temp = " ".join(line_list)
    self.lstOnHand[self.c][0] = temp
    self.c += 1

self.rowName = "amount"
self.query.execute("select {rn} from {tn};".format(rn = self
   .rowName, tn = self.tableName))
```

```
self.row = self.query.fetchall()
self.c = 0
for line_list in self.row:
    temp = "".join(str(line_list))
    temp1 = temp.split('(')
    temp2 = temp1[1].split(',')
    self.lstOnHand[self.c][1] = temp2[0]
    self.c += 1
self.rowName = "units"
self.query.execute("select {rn} from {tn};".format(rn = self
   .rowName, tn = self.tableName))
self.row = self.query.fetchall()
self.c = 0
for line_list in self.row:
    temp = " ".join(line_list)
    self.lstOnHand[self.c][2] = temp
    self.c += 1
self.response = ""
self.temp5 = ""
self.temp3 = None
self.numOfIng = [0 for x in range(self.numOfRec)]
for z in range(self.numOfRec):
    for m in range(1,len(self.lst[z])):
        if self.lst[z][m] != None:
            self.numOfIng[z] += 1
for z in range(self.numOfRec):
    self.canMake = 0
    print(self.lst[z][0] + ": ")
    self.ValueHolder = [None for x in range(self.numOfIng[z
       ])]
    self.p = 0
    for l in range(self.numOfIng[z]+1):
        self.temp5 = list(filter(None,self.lst[z]))
        for m in range(len(self.temp5)):
            if self.lst[z][m] != None:
                self.temp3 = self.lst[z][m].split(" ")
                if len(self.temp3) > 1:
                    #temp4 = the name of the ingredient
                    self.temp4 = ""
                    #temp5 = the entire row of the
                       ingredient
                    self.temp5 = ""
                    self.temp5 = list(filter(None,self.temp3
                       ))
                    for n in range(2,len(self.temp5)):
                        self.temp4 += self.temp5[n] + " "
                    #for r in range(self.numOfIng[z]):
                    if l < len(self.lstOnHand) and self.
                       lstOnHand[l][0] != None and self.
                       temp4 == self.lstOnHand[l][0] + " "
                       and self.temp5[0] < self.lstOnHand[l
                       ][1]:
```

```
                    #print(self.lst[z])
                    self.canMake += 1
                    print("Have enough " + self.temp4 +
                        "on hand")
                    self.t = abs(float(self.lstOnHand[l
                        ][1]) - float(self.temp5[0]))
                    print("You'll be left with: " + str(
                        self.t) + " " + self.lstOnHand[l
                        ][2])
                    self.ValueHolder[self.p] = self.lst[
                        z].pop(m)
                    print(self.ValueHolder[self.p])

                    print(str(self.canMake) + " " + str(
                        self.numOfIng[z]))
                    if self.canMake != 0 and self.
                        canMake == self.numOfIng[z]:
                         print("Can make " + self.lst[z
                             ][0])
                         print("Would cost: ")
                         self.cost = [None for x in range
                             (self.numOfIng[z])]
                         self.rowName = "cost"
                         self.tableName = "
                             log_purchasehistory"
                         self.query.execute("select {rn}
                             from {tn};".format(rn = self.
                             rowName, tn = self.tableName)
                             )
                         self.row = self.query.fetchall()
                         self.c = 0
                         for value_list in self.row:
                             self.tempC = ", ".join(str(v
                                 ) for v in value_list)
                             if self.c == self.numOfIng[z
                                 ]:
                                  break
                             self.cost[self.c] = float(
                                 self.tempC)
                             self.c += 1
                             self.costPerRec = 0.0
                             self.price = 0.0
                         for v in range(len(self.
                             ValueHolder)):
                             if self.ValueHolder[v] !=
                                 None:
                                  self.factor = float(self
                                      .cost[v])/float(self.
                                      lstOnHand[v][1])
                                  self.moderator = .1 *
                                      self.factor
```

```
                                    self.costPerRec += (self
                                        .factor) * float(self
                                        .ValueHolder[v][0])
                                    self.price += (self.
                                        factor + self.
                                        moderator) * float(
                                        self.ValueHolder[v
                                        ][0])
                                    print("$" + str(self.
                                        costPerRec))
                                    print("Should charge: $"
                                          + str(self.price) +
                                         " per a unit")
                    elif self.inRec(self.lst, self.temp4,
                       len(self.lst)):
                        print(self.temp4 + "is another
                           recipe")
                        self.lst[z].pop(m)
                        break
for z in range(len(self.day1)):
    self.temp = self.day1[z].split(" ")
    for n in range(1,len(self.temp)):
        self.temp2 += self.temp[n] + " "
    for l in range(len(self.lst)):
        if self.temp2 == self.lst[l][0] + " ":
            print(self.temp2)
for z in range(len(self.day2)):
    self.temp = self.day2[z].split(" ")
    self.temp2 = ""
    for n in range(1,len(self.temp)):
        self.temp2 += self.temp[n] + " "
    for l in range(len(self.lst)):
        if self.temp2 == self.lst[l][0] + " ":
            print(self.temp2)
for z in range(len(self.day3)):
    self.temp = self.day3[z].split(" ")
    self.temp2 = ""
    for n in range(1,len(self.temp)):
        self.temp2 += self.temp[n] + " "
    for l in range(len(self.lst)):
        if self.temp2 == self.lst[l][0] + " ":
            print(self.temp2)
for z in range(len(self.day4)):
    self.temp = self.day4[z].split(" ")
    self.temp2 = ""
    for n in range(1,len(self.temp)):
        self.temp2 += self.temp[n] + " "
    for l in range(len(self.lst)):
        if self.temp2 == self.lst[l][0] + " ":
            print(self.temp2)
for z in range(len(self.day5)):
    self.temp = self.day5[z].split(" ")
```

```
            self.temp2 = ""
            for n in range(1,len(self.temp)):
                self.temp2 += self.temp[n] + " "
            for l in range(len(self.lst)):
                if self.temp2 == self.lst[l][0] + " ":
                    print(self.temp2)
        for z in range(len(self.day6)):
            self.temp = self.day6[z].split(" ")
            self.temp2 = ""
            for n in range(1,len(self.temp)):
                self.temp2 += self.temp[n] + " "
            for l in range(len(self.lst)):
                if self.temp2 == self.lst[l][0] + " ":
                    print(self.temp2)
        for z in range(len(self.day7)):
            self.temp = self.day7[z].split(" ")
            self.temp2 = ""
            for n in range(1,len(self.temp)):
                self.temp2 += self.temp[n] + " "
            for l in range(len(self.lst)):
                if self.temp2 == self.lst[l][0] + " ":
                    print(self.temp2)
main()
```

Listing 4.2: Main Code

4.3 Django Models

4.3.1 Recipe Model

```
from django.db import models

# Create your models here.

class recipe(models.Model):
    OZ = "ounces"
    LB = "pounds"
    G = "grams"
    KG = "kilograms"
    TSP = "teaspoon"
    TBSP = "tablespoon"
    PAST = "pastries"
    SERV = "servings"
    SLIC = "slices"
    YIELD_CHOICES = (
        (OZ, "ounce(s)"),
        (LB, "pound(s)"),
        (G, "gram(s)"),
        (KG, "kilogram(s)"),
        (TSP, "teaspoon(s)"),
        (TBSP, "tablespoon(s)"),
        (PAST, "pastry(ies)"),
        (SERV, "serving(s)"),
        (SLIC, "slices")
    )
    JN = "January"
    FB = "February"
    MCH = "March"
    AP = "April"
    MA = "May"
    JU = "June"
    JLY = "July"
    AUG = "August"
    SPT = "September"
    OCT = "October"
    NVM = "November"
    DEC = "December"
    NONE = "None"
    Season = (
        (JN, "January"),
        (FB, "February"),
        (MCH, "March"),
        (AP, "April"),
        (MA, "May"),
        (JU, "June"),
        (JLY, "July"),
        (AUG, "August"),
        (SPT, "September"),
        (OCT, "October"),
        (NVM, "November"),
        (DEC, "December"),
        (NONE, "None")
    )
    SeasonUntil = (
        (JN, "January"),
        (FB, "February"),
        (MCH, "March"),
        (AP, "April"),
        (MA, "May"),
        (JU, "June"),
        (JLY, "July"),
        (AUG, "August"),
        (SPT, "September"),
        (OCT, "October"),
        (NVM, "November"),
        (DEC, "December"),
        (NONE, "None")
    )
    HR = "Hour"
    SEC = "Second"
    MIN = "Minute"
    timeStamps = (
        (HR, "Hour"),
        (SEC, "Second"),
        (MIN, "Minute")
    )
    recName = models.CharField(max_length = 100)
    amount = models.FloatField(default = 0)
    units = models.CharField(
        max_length = 20,
        choices = YIELD_CHOICES,
        default = LB,
    )
    season = models.CharField(
        max_length = 20,
        choices = Season,
        default = JN,
    )
    seasonUntil = models.CharField(
        max_length = 20,
        choices = SeasonUntil,
        default = JN,
    )
    timeToMake = models.FloatField(default = 0)
    timeToMakeUnits = models.CharField(
        max_length = 10,
        choices = timeStamps,
        default = HR,
    )
    ingredients = models.TextField()
    def __str__(self):
        return self.recName
```

Listing 4.3: Recipe Model

4.3.2 Log Model

```
from django.db import models

# Create your models here.

class PurchaseHistory(models.Model):
    OZ = "ounces"
    LB = "pounds"
    G = "grams"
    KG = "kilograms"
    TSP = "teaspoon"
    TBSP = "tablespoon"
    PAST = "pastries"
    SERV = "servings"
    SLIC = "slices"
    YIELD_CHOICES = (
        (OZ, "ounce(s)"),
        (LB, "pound(s)"),
        (G, "gram(s)"),
        (KG, "kilogram(s)"),
        (TSP, "teaspoon(s)"),
        (TBSP, "tablespoon(s)"),
        (PAST, "pastry(ies)"),
        (SERV, "serving(s)"),
        (SLIC, "slices")
    )
    ingredient = models.CharField(max_length = 100)
    amount = models.FloatField(default = 0)
    units = models.CharField(
        max_length = 20,
        choices = YIELD_CHOICES,
        default = LB,
    )
    cost = models.FloatField(default = 0)
    day = models.IntegerField(default = 0)

    def __str__(self):
        return self.ingredient

class customerHistory(models.Model):
    name = models.CharField(max_length = 32, default = None)
    bakedGood = models.TextField(default = None)
    YNG = "0 -> 12"
    TEEN = "13 -> 19"
    YNG_AD = "20 -> 29"
    AD = "30 -> 59"
    SNR = "60 -> 120"
    AGE_RANGE = (
        (YNG, "0 -> 12"),
        (TEEN, "13 -> 19"),
        (YNG_AD, "20 -> 29"),
        (AD, "30 -> 59"),
        (SNR, "60 -> 120")
    )
    AgeRange = models.CharField(
        max_length = 20,
        choices = AGE_RANGE,
        default = YNG_AD,
    )
    day = models.IntegerField(default = 1)

    def __str__(self):
        return self.name

class sales(models.Model):
    bakedGood = models.CharField(max_length = 32, default = None)
    amtSold = models.IntegerField(default = 0)
    amtMade = models.IntegerField(default = 0)
    day = models.IntegerField(default = 1)

    def __str__(self):
        return self.bakedGood

class resourcesUsed(models.Model):
    OZ = "ounces"
    LB = "pounds"
    G = "grams"
    KG = "kilograms"
    TSP = "teaspoon"
    TBSP = "tablespoon"
    PAST = "pastries"
    SERV = "servings"
    SLIC = "slices"
    YIELD_CHOICES = (
        (OZ, "ounce(s)"),
        (LB, "pound(s)"),
        (G, "gram(s)"),
        (KG, "kilogram(s)"),
        (TSP, "teaspoon(s)"),
        (TBSP, "tablespoon(s)"),
        (PAST, "pastry(ies)"),
        (SERV, "serving(s)"),
        (SLIC, "slices")
    )
    ing = models.CharField(max_length = 32, default = None)
    amt = models.FloatField(max_length = 5, default = None)
    units = models.CharField(
        max_length = 20,
        choices = YIELD_CHOICES,
        default = LB,
    )

    day = models.IntegerField(default = 1)

    def __str__(self):
        return self.ing
```

Listing 4.4: Log Model

Chapter 5

Discussion and Future Work

5.1 Summary of Results

Looking over what this project has accomplished so far, it sheds light on an intellectually provocative idea; a way for bakers to manage their business more effectively in these times with scarce resources. However, due to my lack of strength in the actual coding, I believe that this is not shown in the results. What the system can do, is only a fraction of what the system was designed to do. Specifically what has been accomplished is able to parse through all of the database, and assess whether a recipe can be made or not. It also does the pricing for that recipe. From an analysis of the logical progression of the system, this seams like an easy task to then use CPSO, however, due to time constraints and a busy schedule leaves this to be a very difficult task.

5.2 Future Work

The future work for this project is a long list of hopes for solving an MSCRP styled problem. This project will need a lot of work on it to make it successful. Building from the ground up, I believe that it may be better to create new python code than fix the existing code. Some of the structure for the code I disagree with how I built it all those months ago, because of the problems I am having now with the system. Since it was so long ago, I cannot remember the reasoning for all of the decisions that I made. Also, I believe that I fell into a pit trap with feeling the need to keep the old code because of the work that I had already put in, and the work that would need to be redone to have success. Looking to future functionality of the system, there are still a lot to be done. Every single factor about a day influences how people behave, and then how they decide to spend their money. Therefore, I would like to add on further detail about the day to figure out trends, and how one could use that data to make a better menu. This not only benefits the bakery, it also benefits the customers. Providing consumers with what fits best in their day will make them feel better, and make sure that they can enjoy whatever baked good is put on the shelves. Very few people will walk into bakeries without the intention of buying food, and therefore this is also wasted time for them if they go in there and buy nothing. Another feature would be creating an intuitive and fluid schedule that would allow for there to be more interactive baking plans. This would help bakers be able to setup for the following week, and plan more efficiently. As mentioned earlier, the smoother a bakers plan the better. Because of the rigorous work that needs to go into maintaining a bakery, there are other features that would be highly beneficial, such as tracking health and safety codes, fire codes, etc. All of these are processes that need to happen by a scheduled time, and therefore can keep the bakery out of legal problems.

5.3 Conclusion

Throughout this research, there has been one central problem to be solved. This problem is that of bakeries, and keeping them in business. Although there are more factors going into a bakery than just what they bake, such as expenses and legal matters, through making this process easy for the bakery, it can allow them to spend time doing what they've been training to do. To become the best baker that they can be.

Bibliography

[1] Cakeboss.

[2] Django version 2.0, 2017.

[3] Marjan Abdeyazdan. A new method for the informed discovery of resources in the grid system using particle swarm optimization algorithm (rdt_pso). *The Journal of Supercomputing*, 73(12):5354–5377, 2017.

[4] Christopher W Cleghorn and Andries P Engelbrecht. A generalized theoretical deterministic particle swarm model. *Swarm intelligence*, 8(1):35–59, 2014.

[5] U Diwekar. Introduction to applied optimization, volume 80 of applied optimization, 2003.

[6] Veysel Gazi and Kevin M. Passino. *Particle Swarm Optimization*, pages 251–279. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[7] Veysel Gazi and Kevin M Passino. *Swarm stability and optimization*. Springer Science & Business Media, 2011.

[8] Tony Hauber and Bartek Gorny. Django-schedule, 2017.

[9] Pinar Hosafci. Global growth: It's not easy, Sep 2016.

[10] Bassem Jarboui, Najeh Damak, Patrick Siarry, and A Rebai. A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Applied Mathematics and Computation*, 195(1):299–308, 2008.

[11] Jhi-Young Joo and Marija D Ilić. Multi-layered optimization of demand resources using lagrange dual decomposition. *IEEE Transactions on Smart Grid*, 4(4):2081–2088, 2013.

[12] R Kennedy. J. and eberhart, particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks IV, pages*, volume 1000, 1995.

[13] Li-lan Liu, Zhi-song Shu, Xue-hua Sun, and Tao Yu. Optimum distribution of resources based on particle swarm optimization and complex network theory. *Life System Modeling and Intelligent Computing*, pages 101–109, 2010.

[14] Daniel J. Power and Ramesh Sharda. *Decision Support Systems*, pages 1539–1548. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[15] Yuhui Shi and Russell C Eberhart. Empirical study of particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages 1945–1950. IEEE, 1999.

[16] Bernhard Thalheim. *Dependencies in relational databases*. Springer-Verlag, 2013.

[17] Timothea Xi. Bakery industry analysis, 2013.