Restaurant Recommendation System Using The Artificial Bee Colony Algorithm

Vijay Bajracharya
RIT
Rochester, NY 14623
vb6487@rit.edu

Nathan Borkholder
RIT
Rochester, NY 14623
nrb4824@rit.edu

Deepak N. Talapaneni
RIT
Rochester, NY 14623
dt3237@rit.edu

Abstract

One of the major reasons diners struggle to pick a restaurant to eat at is the sheer number of establishments offering a variety of cuisines, services, and ambience. Existing systems rely on traditional filtering techniques to recommend restaurants to diners but this requires the diners to have strict and exact preferences which leaves no leeway. Instead, this paper focuses on relaxing these requirements by using a weighting system for categories such as price, locality, cleanliness etc. and then using a novel discretized variant of the Artificial Bee Colony algorithm to maximize user satisfaction and recommend restaurants to the users. Evaluations show that the recommendation model can accurately recommend restaurants that best characterizes the user's preferences without need for excessive filtering.

1 Introduction

The restaurant industry in the United States has seen a massive growth in terms of the number of establishments serving foods from all over the world. These establishments offer a wide variety of services, cuisines, and environments. As such, for a diner, the decision-making process for picking a restaurant has become increasingly challenging. The overwhelming number of choices and the various permutations of services make choosing a restaurant a stressful task.

Although there are a plethora of existing systems that are capable of recommending restaurants to users, they often use filtering techniques to narrow down the search. While this may be useful for a diner confident in their preferences, most of the times, the preferences can be more subjective in nature. Suggesting restaurants when there is no clear preference has to be more subtle than database filtering. As such, developing an accurate and effective recommendation model without the need for explicit filtering of data is essential for generating useful restaurant recommendations.

This paper proposes a novel restaurant recommendation model based on the Artificial Bee Colony (ABC) algorithm. The ABC algorithm is a meta-heuristic swarm optimization algorithm inspired by the foraging behavior of bees. It is often used for solving complex optimization problems on continuous function space. This application study will explore the problem-specific discretization of the ABC algorithm specifically targeted at recommending a restaurant that best characterizes a user's preferences.

1.1 Motivation

A recommendation system solves decision making problems by suggesting content based on a user's preferences and behaviors. This can be useful when there are an overwhelming number of choices and

the user is looking for a personalized subset to choose from. This is a frequent point of indecisiveness for individuals or groups looking to dine. This problem arises due to varying food-related preferences and the sheer amount of restaurants to choose from. In such a situation, a recommendation system can provide a short list of restaurants by optimizing for a user preferences.

Our decision to leverage swarm intelligence and in particular, the Artificial Bee Colony (ABC) algorithm stems from its ability to distribute the workload. Each agent will contribute to processing the information in an efficient manner. One study showed that the ABC algorithm leads to better clustering accuracy while also being able to handle local optimas [5]. Another study showed that the ABC algorithm enhanced auxiliary material suggestions on Facebook by providing more accurate recommendations to its users [2].

1.2 Background

1.2.1 Metaheuristic Optimization Algorithms

Metaheuristic shares a similar definition to heuristic whereby heuristic usually stands for approximation or educated guessing whereas metaheuristic, while still being an approximation, is more generalizable to a broader set of problems. It usually involves an algorithmic framework to optimize some function by way of intensification and diversification. Intensification is the process of honing in on a local trend while diversification is the process of exploring alternative regions for better solutions. The combination of these two components make up a metaheuristic algorithm.

A metaheuristic optimization algorithm uses a metaheuristic to solve optimization problems. These optimization problems generally involve maximizing or minimizing an objective function based on the assumptions and constraints of the problem. A metaheuristic optimization algorithm can be viewed as an iterative process, a self-organizing system, or an ideal algorithm such as the Newton-Raphson method.

Most metaheuristic optimization algorithms have a notion of intensification and diversification. These two operations are usually in *tension*, meaning that improving one will downgrade the other.

- Intensification involves honing in on a local region to find a strictly better solution.
- Diversification involves expanding the search space with no regard to local information.

A good metaheuristic optimization algorithm will aim to balance the two.

1.2.2 Swarm Algorithms

A swarm algorithm is a concept where a group of decentralized, self-organized systems, acting only locally with a simple set of individual rules, yields a collective global intelligence. These algorithms are based off the Boids model which simulated the flocking behavior of birds [7]. In this model, there are a population of birds and each bird has three behaviors:

- Separation: steer to avoid crowding local flockmates
- Alignment: steer towards the average heading of local flockmates
- Cohesion: steer to move toward the center of mass

Using these three simple rules, the program is able to simulate the flocking nature of birds. Modern swarm algorithms operate in a similar manner. The movement of each agent can be defined using a set of equations and during each iteration, the agent will move based on their current state. While simple in concept, this leads to a collective intelligence that allows the agents to explore a large area and also find good solutions.

1.2.3 Foraging Behavior of Bee Colonies

To study the foraging behavior of honeybee colony, researcher Tereshko, setup a model consisting of three components: food sources, employed foragers, and unemployed foragers [8]. Each of the components can be described as follows:

- Food Sources: Each food source contains a nectar amount that determines the richness of the food source. Based on the nectar amount, the quality of the food source as well as its likelihood of being recruited is established[3]
- Employed Foragers: The *Employed* bees are recruited to a particular food source in the environment and relay neighboring information to other bees regarding the quality of the food source [3].
- Unemployed Foragers: Unemployed foragers can be divided into *Onlookers* and *Scouts*. The *Onlookers* find a food source to recruit based on information supplied by an employed bee. In contrast, the *Scout* finds a food source by moving randomly through the environment [3].

An important aspect of the foraging behavior is defined by an information sharing technique called the *waggle dance*. The *waggle dance* is a behavior of the *Employed* bees that informs the *Onlooker* bees regarding the richness of the neighborhood[4]. The *Onlooker* bees observe all of the *waggle dances* to determines which food source they direct themselves towards.

1.3 Challenges

The core challenge that this application study faces is the discrete solution space that the ABC algorithm has to operate within. All of the equations defined in the original paper of the ABC algorithm [3] are designed for a real and continuous function space. Adapting these equations to not only a discrete function space but also handling categorical and mixed data-types are the main challenges for this paper. Additionally, an objective function must also be characterized such that it is specific to a user's preferences.

2 Related Work

Recommendation systems have been increasing in popularity with the sharp increase in information readily available to the public. These systems help streamline a user's experience and avoid confusion when searching for information[5]. Often for tourists recommendation systems are indispensable as they help them find activities and hotels based on a number of criteria[1]. Many models struggle to effectively account for multiple criteria in the way a recommendation system does. In a recent study for movie recommendations, the authors proposed a simple 3 step process to properly recommend movies to their users. First they would preprocess users information and transform it into a usable form. Secondly they would extract attributes from the collection based on users qualities. For this step they chose to use the Artificial Bee colony algorithm for speed and accuracy. Finally they would recommend movies based on the users ratings and other similar users ratings[5]. It is important to recognize the choice of the ABC algorithm here. In a study conducted by Facebook, the ABC algorithm was found to be an effective way to obtain better solutions than other methods and is capable of obtaining a near-optimal solution in a reasonable time[2].

3 Problem Description

3.1 Maximizing User Satisfaction

The problem that the restaurant recommendation model is to trying to solve is to recommend a suitable restaurant using the Artificial Bee Colony (ABC) algorithm based on the following user preferences:

- Price
- Distance
- Cleanliness and Hygiene
- Locality/Neighborhood
- Wait Times
- Portion Sizes
- Overall Restaurant Rating

Given a list of restaurants $\vec{x} = (x_1, x_2, ..., x_{n-1}, x_n)$ where a restaurant r_i is defined by categories mentioned above and a survey of the user's preferences as input, the proposed solution must maximize an objective function $f_i(\vec{x})$ where the objective function dynamically changes based on the user's preferences and is a measure of user satisfaction.

A recommendation system fits into the generalized framework of a global optimization problem:

$$\begin{array}{lll} \text{maximize} & f_i(\vec{x}) & \text{where} \quad i = 1, 2, \ldots, N \\ \text{subject to} & constraints & \end{array}$$

For the recommendation system, the objective function $f_i(\vec{x})$, must be constructed from some combination of the user's preferences for the aforementioned categories. The constraints must ensure that the recommended restaurants must adhere to hard rules such as cuisine choices. The output must be a restaurant name that captures the user's preferences.

3.2 Discretization of the ABC algorithm

The ABC algorithm is designed for problems in a continuous solution space. As such the proposed solution must discretize the three main behaviors of the ABC algorithm:

3.2.1 Employed Bee

An employed bee generates a candidate solution using the following equation:

$$v_i = x_i + \phi_i * (x_i - x_j) \tag{1}$$

where, v_i is the new candidate solution, x_i is the position of the employed bee i, ϕ_i is a random number sampled between $[-1, 1]$, and x_j is a randomly selected member of the population. This equation must be discretized such that the new candidate solution is a valid restaurant between two restaurants.

3.2.2 Onlooker Bee

The onlooker bee generates a probability for each food source depending on quality of the food source (higher quality food sources being more preferable). Then it recruits a neighboring food source using Eqn. (1). This step also needs to be discretized such that the neighboring food source is present in the list of restaurants.

3.2.3 Scout Bee

When a food source is abandoned due to the quality of the neighboring food sources being poor, the scout bee will move to a random food source using the following equation:

$$x_i = lower\_bound + \phi_i * (lower\_bound - upper\_bound) \tag{2}$$

where, x_i is the position of the new food source, ϕ_i is a random number between $[0, 1]$ and the $lower\_bound$ and $upper\_bound$ are problem specific. There is no quantifiable notion of lower bound and upper bound for a list of restaurants so the random food source will have to be sampled using a different metric.

4 Proposed Solution

Our approach makes restaurant recommendations by modifying key components of the ABC algorithm (employed bee, onlooker bee, and scout bee phases) that allow it to optimize on discrete categorical data. At it's core, the algorithm initializes the bees at random restaurants and each bee in the population, based on whether it is employed, onlooker, or scout makes actions that explore the neighborhood. The fitness of each of the explored restaurants are calculated using a dynamically defined objective function based on the user's preferences. The restaurant with the highest fitness out of all the explored restaurants is memorized until the algorithm meets the termination criteria and this restaurant is reported to the user as its recommendation.

4.1 Pseudocode

Data: Objective Function (f), Swarm Size (N), Constants: ($abandonment\_limit(L)$)
Result: Global Best Restaurant g^*
Algorithm
- Initialize population at random restaurants and employ, x
- $fitness = f(x_i)$, for selected resturants
- $noImprovement$ = [0 for x_i in x]
- g^* = $x[argmax(fitness)]$
- **while** *(Criterion)* **do**
  - # employed bee phase
  - **foreach** x_i *in population* **do**
    - $neighbor = getNeighbor()$
    - **if** *neighbor.fitness* > $fitness[x_i]$ **then**
      - $x_i = neighbor$
    - **else**
      - $noImprovement[x_i] = noImprovement[x_i] + 1$
    - **end**
  - **end**
  - # onlooker bee phase
  - $fitness\_sum = sum(fitness)$
  - **foreach** x_i *in population* **do**
    - **if** *rand()* < $x_i.fitness/fitness\_sum$ **then**
      - $neighbor = getNeighbor()$
      - **if** *neighbor.fitness* > $fitness[x_i]$ **then**
        - $x_i = neighbor$
      - **else**
        - $noImprovement[x_i] = noImprovement[x_i] + 1$
      - **end**
    - **end**
  - **end**
  - # scout bee phase
  - **foreach** x_i *in population* **do**
    - **if** $noImprovement[x_i] > L$ **then**
      - $x_i =$ random not visited restaurant
    - **end**
  - **end**
  - Update global best g^*
- **end**
- Return g^*

4.2 Defining The Food Source

In the ABC algorithm, the candidate solutions are defined as food sources where the nectar amount represents the desirability of the food source which has cascading effects in how the bees operation. For our model, we have defined a food source to be a restaurant and each restaurant is in itself defined

as follows:

$$\vec{r_i} = [\text{price}, \text{distance}, \text{cleanliness}, \text{locality}, \text{wait times}, \text{portion sizes}, \text{overall rating}]$$

4.3 Data Transformation

Before optimization can be performed, each category that defines a restaurant must be normalized onto the same scale. To do this, we take the data for each restaurant and rate it on a scale of 1-100 depending on its desirability. For example, a rating of 100 for price, would be that the price for that restaurant is the most desirable. In contrast, a rating of 1 for price would mean that the price of foods at that restaurant is undesirable. The same scale applies to each of the 7 categories mentioned above.

4.4 Input and Output

The algorithm begins with an interface that prompts the user to input their preferences for each of the criteria that define a restaurant. Each preference is an integer between 1 and 10 and it forms a vector $\vec{p}$ with the same dimensions as a restaurant. This step is crucial as it tailors the objective function to user-specific needs. These preferences are weights that will determine the richness of a food source. The user inputs are validated to ensure they meet the expected format and range. Correct input is essential for the subsequent phases of the algorithm to function correctly. The algorithm performs the optimization step and produces the name of the restaurant as its final output to the user.

4.5 Objective Function

The objective function is central to the ABC algorithm, serving as the criterion for evaluating the quality of the restaurant found by the bees. It is defined based on the preference ratings provided by the user during the initial prompt. The objective function is constructed as linear combination of user preferences and the corresponding field for that restaurant. For example, for a restaurant $\vec{r_i}$, if $\vec{r_i} = [5, 10, 15, 20, 25, 30, 35]$ and the user preference is $\vec{p} = [5, 5, 2, 2, 3, 3, 10]$ then the fitness of the restaurant is given by $\vec{r_i} \cdot \vec{p}$.

During the search process, each bee evaluates its position (and potential new positions) using this function. The fitness scores guide the bees toward areas in the solution space that offer better outcomes.

4.6 Primary Preference Based Neighborhood Exploration

In our approach, neighborhood exploration is tailored around a primary preference, which is a user-defined criterion determined during the initial prompt. The primary preference is a parameter that signifies the category that user regards as their most important category. The primary preference shapes the search focus, guiding the bees in the solution space.

The neighborhood exploration phase utilizes a window-based mechanism which replaces the equation shown in Eqn.(1) to determine neighboring candidate solutions. The mechanism for this strategy can be described as the following:

- **Window Size:** The exploration window is set to +/- 4 units around the bee's current position indexed by the primary preference. For example, if a bee's index is 50 based on the primary preference, positions from index 46 to 54 are considered.
- **Search Range:** This window defines the potential neighbors' range, ensuring a balance between exploring broadly and exploiting promising areas in detail.
- **Selection Criteria:** Potential neighbors are evaluated within this window based on their fitness values, calculated via the objective function which considers the primary filter and other relevant factors.

The primary preference significant influences the exploration process. By centering the search around a key attribute, the algorithm intensively explores the most relevant areas of the solution space,

enhancing the optimization towards user-prioritized criteria. This targeted approach improves the search efficiency and effectiveness by focusing efforts on areas most valued by the user, thus aligning the algorithm's outputs with user expectations.

4.7 Employed Bee Phase

Employed bees explore near their current position. They attempt to find a better solution by perturbing their current position within a defined max_distance and max_standard_deviation, which keep the search localized but flexible. It uses the primary preference based neighborhood exploration to make these perturbations and it ensures that the candidate solutions are always valid restaurants in discrete space. After exploring, each employed bee evaluates the fitness of the new position. If the new position offers a higher fitness than the current one, the bee moves to this new position.

4.8 Onlooker Bee Phase

Onlooker bees select restaurants based on their fitness levels. Employed bees perform their waggle dance and inform the Onlooker bees regarding the quality of their restaurant. Restaurants with higher fitness have a higher probability of being chosen, which is calculated based on the fitness of the restaurant relative to the total fitness of all restaurants as shown in Eqn.(3).

$$P_i = \frac{f(r_i)}{\sum_j^N f(r_j)} \tag{3}$$

Upon selecting an restaurant, the onlooker bee looks for potential improvements around the neighborhood of the restaurant, guided by parameters like 'max_onlooker_distance'. This phase focuses on exploiting the promising areas identified by the employed bees.

4.9 Scout Bee Phase

If certain solutions cannot be improved further over several iterations, the food source is abandoned and the scout bees are activated. These bees are responsible for random searches, recruiting restaurants randomly across the entire solution space. Eqn.(2) is replaced by a masking mechanism that chooses a random restaurant from the database provided that the restaurant has not been visited before. This mechanism prevents the algorithm from getting stuck in local optima and encourages exploration of new areas.

4.10 Termination Criteria

Unlike known continuous functions, it is not possible to know the ground truth optima for every configuration of the objective function. As such, the optimization occurs iteratively until a max number iterations is reached. Then, the global best resturant is recommended to the user.

5 Implementation

The algorithm was implemented and tested in python using libraries such as $numpy$ and $pandas$ across two Python modules. Employed bees, Onlooker bees, and Scout bees are defined in $bee\_class.py$. The main optimization algorithm is implemented within $abc\_model.py$ which orchestrates the overall algorithmic process, including user interaction, data handling, and the operational phases of the bees. The analogous continuous implementation can be located in $abc\_model\_continuous.py$. All of the datasets are present in the $data$ folder in the repository and this includes the evaluation datasets as well as the full restaurant dataset. Problem specific parameters such as max_distance and max_standard_deviations can be adjusted based on the problem's specifics or user requirements, offering flexibility in how the algorithm explores and exploits the solution space.

6 Evaluation

6.1 Experimental Setup

To evaluate the performance of the proposed solution, we first implemented the continuous version of the ABC algorithm as detailed in [3]. We tested the continuous ABC algorithm on four benchmark functions - rosenbrock function, easom function, ackley function, and the eggcreate function. For each of these functions, we evaluated the mean and standard deviation of the optima over 30 trials.

Then, the performance of the recommendation model was tested using several small and carefully crafted datasets as well as a large restaurant dataset containing a total of 1277 restaurants [6]. While the names of the restaurants belong to actual restaurants in the real-world, the fields corresponding to their prices, rating, etc. were randomly generated. These crafted datasets range in difficulty easy, medium, and hard based on the size of the dataset and similarity between restaurants.

The goal of the crafted datasets are to test the accuracy of the recommendation model. Given, scenarios with objectively correct recommendations, we can determine if the recommendation given by the model accurately characterizes the user's preferences. The large restaurant dataset mimics the number of restaurants that may be available in a metropolitan area and it tests the ability of the model to scale to large datasets while still producing accurate recommendations.

6.2 Results

6.2.1 ABC algorithm Performance

The results for the continuous ABC algorithm on the 4 benchmark function are as follows:

	Benchmark	$f(x)_{min}$	$f(x)_{min}$	$f(x)_{min}$	$f(x)_{min}$
1	Rosenbrock	5.005e-7 $\pm$ 4.222e-7	0.0003238 $\pm$ 0.00030033	2.373e-6 $\pm$ 4.333e-6	0.0010360 $\pm$ 0.0011905
2	Easom	-0.0127796 $\pm$ 0.0	-0.0127796 $\pm$ 0.0	-0.01277964 $\pm$ 0.0	-0.0127796 $\pm$ 0.0
3	Ackley	4.441e-16 $\pm$ 0.0	0.0003332 $\pm$ 0.0002580	4.441e-16 $\pm$ 0.0	0.0004358 $\pm$ 0.0004439
4	Eggcrate	3.660e-153 $\pm$ 9.364e-153	1.179e-12 $\pm$ 4.164e-12	5.425e-152 $\pm$ 2.953e-151	1.672e-14 $\pm$ 3.758e-14
		$N = 100, L = 50$	$N = 100, L = 10$	$N = 30, L = 50$	$N = 30, L = 10$

Table 1: Mean and standard deviation of ABC algorithm over 30 trials on benchmark functions

Across all benchmark functions the abandonment limit was the most impactful in altering the mean and standard deviation. Changing the abandonment criteria from 50 to 30 increased the mean and standard deviations in most cases by multiple orders of magnitude. The higher the population the better the results, however changing the population had minimal impact on the mean and standard deviation. In the case of the ackley function changing the population had no impact on the results, while changing the abandonment criteria changed the results significantly.

6.2.2 Restaurant Recommendation Model Performance

Easy Dataset. The easy dataset shown in Fig.1 is characterized by a small list of 11 restaurants that have the same ratings for all categories except one category. In addition, there are no ties between restaurants and there is a clear restaurant that maximizes user satisfaction even if the user has the same preference over all the categories.

We ran this dataset through the restaurant recommendation model over 30 trials with a population size of 2 bees and an abandonment limit of 5. The result from Table. 1 shows that the recommended restaurant always converges to **res6** which is the expected behavior since **res6** has the best ratings among all categories. This demonstrates that the model can accurately locate the correct restaurant for the easy dataset.

Medium Dataset. The medium dataset shown in Fig.2 is characterized by a list of 20 restaurants with multiple degrees of freedom. However, there are no ties and there is still a restaurant that maximizes user satisfaction even if the user has the same preference over all the categories.

Restaurant Name	Price	Cuisine	Distance	Cleanliness and Hygiene	Locality/Neighborhood	Wait Times	Portion Sizes	Overall Rating
res1	1	Indian	1	1	1	1	1	1
res2	2	Indian	1	1	1	1	1	1
res3	3	Indian	1	1	1	1	1	1
res4	4	Indian	1	1	1	1	1	1
res5	5	Indian	1	1	1	1	1	1
res6	6	Indian	1	1	1	1	1	1
res7	5	Indian	1	1	1	1	1	1
res8	4	Indian	1	1	1	1	1	1
res9	3	Indian	1	1	1	1	1	1
res10	2	Indian	1	1	1	1	1	1
res11	1	Indian	1	1	1	1	1	1

Figure 1: An easy difficulty data-set with only one degree of freedom (Price)

Restaurant Name	Price	Cuisine	Distance	Cleanliness and Hygiene	Locality/Neighborhood	Wait Times	Portion Sizes	Overall Rating
res1	10	Indian	23	23	1	1	1	1
res2	25	Indian	44	13	1	1	1	1
res3	35	Indian	55	23	1	1	1	1
res4	45	Indian	66	45	1	1	1	1
res5	24	Indian	77	56	1	1	1	1
res6	21	Indian	91	44	1	1	1	1
res7	66	Indian	23	13	1	1	1	1
res8	22	Indian	42	34	1	1	1	1
res9	23	Indian	23	23	1	1	1	1
res10	13	Indian	14	85	1	1	1	1
res11	34	Indian	55	23	1	1	1	1
res12	56	Indian	66	13	1	1	1	1
res13	23	Indian	44	45	1	1	1	1
res14	11	Indian	89	73	1	1	1	1
res15	89	Indian	13	13	1	1	1	1
res16	91	Indian	91	91	1	1	1	1
res17	22	Indian	23	65	1	1	1	1
res18	33	Indian	44	13	1	1	1	1
res19	42	Indian	55	23	1	1	1	1
res20	23	Indian	13	45	1	1	1	1

Figure 2: A medium difficulty data-set with three degrees of freedom (Price, Distance, Cleanliness)

We ran this dataset over 30 trials with a population size of 2 bees and an abandonment limit of 5. The result from Table. 2 shows that the recommended restaurant always converges to **res16**. Again this is the expected and correct behavior because **res16** is better than other restaurants in all categories. This demonstrates the model's ability to distinguish between categories even with multiple degrees of freedom.

Hard Dataset. The hard dataset shown in Fig.3 is characterized by a list of 20 restaurants with multiple degrees of freedom. There are ties between restaurants and the "best" restaurant can vary depending on the user's preferences.

We ran this dataset over 30 trails with a population size of 2 bees and an abandonment limit of 5. The results show that when the user preference is equal for all categories, the model recommends **res8** and **res9**. When the user preferred price over distance, the model always converged to **res8** and when the user preferred distance over price, the model always converged to **res9**. This is the correct and expected behavior. Moreover, restaurants that are rated very highly for certain categories but poorly for other categories are rightfully not recommended by the system.

Large Restaurant Dataset. As mentioned earlier, the large restaurant dataset mimics the quantity of the restaurants that may be available in a real city. Results for running the recommendation model on this dataset shows that while the recommendation may not always converge to the same restaurant, it still characterizes the user preferences well. For instance, if the user prefers price over all other categories, the model recommends a restaurant that has a high rating for price. This holds for all other categories as well. Moreover, given an initialization of 100 bees, the model only takes 0.150 seconds to make one recommendation in this dataset.

Restaurant Name	Price	Cuisine	Distance	Cleanliness and Hygiene	Locality/Neighborhood	Wait Times	Portion Sizes	Overall Rating
res1	100	Indian	1	1	1	1	1	1
res2	1	Indian	100	1	1	1	1	1
res3	1	Indian	1	100	1	1	1	1
res4	1	Indian	1	1	100	1	1	1
res5	1	Indian	1	1	1	100	1	1
res6	1	Indian	1	1	1	1	100	1
res7	1	Indian	1	1	1	1	1	100
res8	80	Indian	50	50	50	50	50	50
res9	50	Indian	80	50	50	50	50	50
res10	50	Indian	50	50	50	50	50	50
res11	100	Indian	100	1	1	1	1	1
res12	87	Indian	78	45	23	23	12	5
res13	5	Indian	87	78	45	23	23	12
res14	12	Indian	5	87	78	45	23	23
res15	23	Indian	12	5	87	78	45	23
res16	23	Indian	23	12	5	87	78	45
res17	45	Indian	23	23	12	5	87	78
res18	78	Indian	45	23	23	12	5	87
res19	23	Indian	23	23	23	23	23	23
res20	100	Indian	100	2	2	2	2	100

Figure 3: A hard difficulty data-set

7 Discussion

The ABC algorithm performed remarkably well in the discrete space that was defined. The majority of the algorithm was usable in both continuous and discrete spaces with defining neighbors being the primary change. We initially ran into problems determining what a neighbor was for a bee as no previous papers mentioned how they handled neighborhoods in discrete spaces. Our implementation of neighborhoods does come with higher computation time but is more likely to converge faster. Because of our faster convergence we needed to alter how frequently scout bees would abandon their current food source and find another random food source to prevent getting stuck at local optima.

Preparing the data for the ABC algorithm is a crucial step and one that sets our experiment apart. We wanted to avoid excessive sorting of the database as it is not true to the algorithm. Other algorithms used Euclidean distance comparing multiple user vectors. However, because we are a recommender system that takes only the current users input, and uses a database of ratings to recommend restaurants we needed to handle the data in a unique way. We decided on the approach of pre-sorting by cuisine type and then removing any cuisines that the user didn't request. Then sorting the data by one metric based on the user's input. This slows down the algorithm, but is the only way to guarantee an accurate neighbor relation.

8 Conclusion

This paper presented a novel mechanism for recommending restaurants to users based on weighted preferences of restaurant qualities such as price, cleanliness, distance, locality, wait times, portion sizes, and overall rating. To do so, the model uses a discretized variant of the ABC algorithm. Experiments were conducted on both the continuous and discrete solution space and results show that the ABC algorithm is able to perform well on benchmark functions despite its simple nature. Moreover, the discrete variant is able to accurately recommend restaurants that matches the preferences of the users.

8.1 Future Work

Multi-object function. The restaurant recommendation model currently recommends restaurants based on one user's preference. In practice, it would be more useful if the model tried to optimize for preferences of multiple users. This would require use of a Pareto front which seeks to balance the objective functions of multiple conflicting preferences. This is also common in real life where one diner may have a preference for a certain degree of cleanliness but another diner may not share

the same preferences. In this case, the Pareto front optimization would allows us to recommend a restaurant that is a middle ground between the two conflicting preferences.

Momentum Hyper-parameter. A lot of swarm algorithms have a concept of momentum. For example, in the standard PSO, there is a variant that introduces inertia when calculating the position of the particles. This allows the particle to maintain the momentum of the previous velocity and it tends to lead to better convergence. The ABC algorithm by default does not have a notion of momentum. A parameter such as this could be added to the neighbor calculation during the employed bee phase for potentially better results.

Sophisticated Neighborhood Exploration. To further enhance the robustness of the discrete ABC algorithm, future research may improve the neighborhood exploration by not having to rely heavily on one primary preference. Instead, incorporating a comprehensive multi-preference approach would allow the algorithm to evaluate and integrate all of the categories simultaneously.

References

[1] Saman Forouzandeh et al. "A hotel recommender system for tourists using the Artificial Bee Colony Algorithm and Fuzzy TOPSIS Model: a case study of tripadvisor". In: *International Journal of Information Technology & Decision Making* 20.01 (2021), pp. 399–429.

[2] Chia-Cheng Hsu et al. "A personalized auxiliary material recommendation system based on learning style on Facebook applying an artificial bee colony algorithm". In: *Computers & Mathematics with Applications* 64.5 (2012), pp. 1506–1513.

[3] Dervis Karaboga. "Artificial bee colony algorithm". In: *scholarpedia* 5.3 (2010), p. 6915.

[4] Dervis Karaboga and Bahriye Akay. "A comparative study of artificial bee colony algorithm". In: *Applied mathematics and computation* 214.1 (2009), pp. 108–132.

[5] Rahul Katarya. "Movie recommender system with metaheuristic artificial bee". In: *Neural Computing and Applications* 30.6 (2018), pp. 1983–1990.

[6] *Restaurant Data*. Accessed: September 20, 2024. n.d. URL: https://github.com/jtleek/modules/blob/master/03_GettingData/03_02_summarizingData/data/restaurants.csv.

[7] Craig W Reynolds. "Flocks, herds and schools: A distributed behavioral model". In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, pp. 25–34.

[8] Valery Tereshko. "Reaction-diffusion model of a honeybee colony's foraging behaviour". In: *International conference on parallel problem solving from nature*. Springer. 2000, pp. 807–816.