Introduction to AI Final Project - Meal Planner

Nadav Lederman (ID: 208585760), Mor Ben Ami (ID: 209789924), Daniel Gibor (Bogatyrevich) (ID: 334065281)

I. Introduction

This project addresses the critical challenge of optimizing food inventory management in kitchens through AI-driven planning techniques. Our goal is to minimizing food waste while creating a flexible platform to maximize personal tase, preparation time and shelf time.

Food waste is a pressing global issue, with up to 40% of food in the United States going uneaten. A significant portion of this waste occurs in homes and restaurants, largely due to ineffective inventory management and meal planning. By developing an intelligent system that optimizes ingredient usage, we aim to reduce waste, save costs, and lessen the environmental impact of food disposal.

To navigate the complex decision-making involved in meal preparation and inventory management, we model the problem as a state space search, where each state reflects the current inventory and meal plan. Actions, in this context, are recipe selections that transform the state by consuming ingredients and adding meals to the plan. The goal is to find a sequence of actions that optimizes meal planning based on our evaluation criteria.

To solve this optimization problem, we implement and compare several algorithms:

- Greedy Algorithm: Select the best immediate action at each step.
- Simulated Annealing: Iteratively refines a random solution to improve performance.
- Reinforcement Learning: Learns an optimal policy through repeated interactions with the environment.

Initially we planed on using GRAPHPLAN algorithm as well for solving this, but due to time constraints, and the size of datasets after a number of trials we decided to remove it (we were not able to properly test it since every run took a very long time).

By exploring these diverse techniques, we aim to identify the most effective method for optimizing food inventory usage. To ensure usability, we've developed a user-friendly graphical interface using Python's Tkinter library. Key components of the interface include:

- A Home frame providing an overview and instructions.
- An Upload frame for managing product and inventory data.
- A Settings frame for configuring optimization parameters and selecting solvers.
- A Results frame displaying optimized meal plans, scores, and solver execution times.

This intuitive GUI enables users to interact with the AI system without requiring technical knowledge. Furthermore, our system's flexible architecture allows for future enhancements, such as incorporating nutritional balance and dietary restriction constraints.

During this project we have used a number of LLM's for different use cases, Perplexity for finding previous work done in similar topics, and Claude for assistance with generating initial test data, edits to the report and solving bugs in the code.

II. Previous Work

Several algorithms and approaches have been used to tackle the challenge of optimizing food inventory usage and meal planning. Linear programming has emerged as a particularly useful technique for solving these types of optimization problems. Linear programming was first applied to diet optimization in the 1940s by economist George Stigler, who tried to determine the minimum cost diet that would meet basic nutritional requirements [1]. While Stigler was unable to find an exact solution due to the complexity of the problem, mathematician George Dantzig later developed the simplex algorithm that made solving such linear programming problems feasible [1]. More recently, researchers have applied linear programming to develop nutritionally adequate, low-cost meal plans. For example, the USDA has used diet modeling since 1975 to generate balanced, affordable menus in its Thrifty Food Plan[1]. Linear programming has also been used to design school lunch menus that reduce greenhouse gas emissions and water use [1].

A. Common algorithms used include:

- Standard linear programming to minimize cost while meeting nutritional constraints [1]
- Individual diet modeling to translate nutrient recommendations into personalized food choices [2]
- ε-constraint method to analyze trade-offs between multiple objectives like cost and environmental impact [3]

These are classical algorithms that solve CSP's, but mostly lack the flexibility to expand the problem and allow different metrics to be added easily.

B. Current benchmarks focus on metrics like:

- Minimizing food waste (e.g. achieving 0 grams waste in optimized meal plans) [3]
- Reducing greenhouse gas emissions (e.g. 15-60% reduction compared to baseline diets) [3]
- Lowering diet costs (e.g. 25-30% cost reduction while maintaining nutrition) [1]

We are interested mainly in Minimizing food waste although using our algorithms it would be easy to optimize for all three benchmarks given the data for food items and recipes.

C. Common modeling assumptions include:

- Ignoring food preparation methods and treating all ingredients in a recipe equally[3]
- Assuming nutrients are evenly distributed within food categories [2]
- Using average serving sizes and consumption patterns from dietary surveys [1]
- Allowing small deviations from exact nutritional requirements (e.g. 10% flexibility) [3]

In the next section we discuss our assumptions made in order to model the problem in a way to gives us much flexibility, for both removing self imposed constraints, and relieving assumptions made.

III. Methodology

We modeled our problem using our current "State" as the given food items we can use, their quantities, their expiry date and our current date. And a set of "Actions" defined by recipes, where we track the food items needed, the cooking time, the nutritional value and a rather subjective measure of tastefulness. This structure allows great flexibility both in adding fields to each food item (CO2 emmisions per gram etc.) and to recipes (analysis of nutritional values for making custom diets).

1) Algorithms: We used four different algorithms to solve the problem, each algorithm takes a different approach to find an optimal solution, starting with a naive greedy solution.

2) Greedy Solver: The Greedy Solver makes the best choice at each step without looking ahead. It works in the following manner:

- Checks available actions (meals that can be made with current ingredients)
- Picks the action with the highest score
- Updates the state and repeats until the goal is reached or no more actions are possible

3) Reinforcement Learning Solver: This solver learns from repeated attempts to solve the problem. It uses Q-learning, a model free RL algorithm. The process includes:

- Randomly trying different actions and observing outcomes
- Updating a "Q-table" that stores the expected value of each action in each state
- Balancing exploration of new options with using known good choices (epsilon greedy)

4) Simulated Annealing Solver: A method used for solving CSP's, and is a take of the hill climber algorithm, This solver:

- Starts with a random solution
- Makes small changes to the solution
- Accepts improvements always and worse solutions sometimes, based on a "temperature" that decreases over time

A. Assumptions and Success Criteria

Our algorithms are general solvers for the problem setup, thus we have created a number of different optimization goals for the general solution.

1) All solvers work towards two main goals::

- Minimize waste by using ingredients before they expire
- Maximize or minimize chosen parameters: Shelf time, Taste rating, Number of steps, Preparation time and Number of products

2) Our system makes the following key assumptions::

- We have accurate data on ingredient expiration dates and nutritional values, without expiration date the model is useless since this is our main objective.
- Meal preparation time is not a constraint - it can be used as a parameter for optimizing upon, but is not defined in a way that we won't or will choose a recipe due to preparation time.
- We do not take into account shelf life for future planning of meals, meaning if a recipe has a shelf life of two weeks or one day won't change the meals chosen into the future.

3) Reward functions: Our project uses three problem setups as the infrastructure for solving our objectives, each problem setup is created with a different reward function. All problems have a final score which is the total sum of all action scores chosen along the way.

1) CountExpiredItemsProblem: This function scores meals based on the number of items that would expire if not used:
$$\text{score} = \frac{1}{\text{expired count} + 1}$$
where expired count is the number of items that would expire by the next day if not used in the current meal. This is the most accurate metric for our problem definition, but as we can see, it is discrete in nature. There is no true relation between the number of expired items for one action on day 1 to the number of expired items per action for day 2.
2) MinimizeWasteProblem: This problem scores meals based on how soon their ingredients will expire. It calculates for each action the sum over each ingredient:
$$\text{score} = \frac{1}{\text{expired count}+1} + \sum \frac{1}{\text{expiration date} - \text{current date} + 1}$$
The main goal of this problem setup is to solve the issue with the previous setup, and give a sense of continuity to choices of actions over time. This problem can be thought of as a heuristic for the first problem, and we shall see why. We note that since a legal action is a recipe where all ingredients exist and are not expired the denominator is always positive.
3) ParametersProblem: This problem scores according to recipes parameters, trying to maximize or minimize chosen fields, this problem combined with MinimizeWasteProblem gives us a general problem that plays on the tradeoff between our parameters and the waste.

Claim 1. The MinimizeWasteProblem heuristic serves as an admissible heuristic for CountExpiredItemsProblem.

Proof: Since we are trying to minimize the cost in this problem, for admissibility we must show MinimizeWasteProblem is larger than CountExpiredItemsProblem for every action and state. But by definition it is defined as a sum of CountExpiredItemsProblem score, and a positive value. ■

For evaluation we made a number of datasets of both products and recipes, ranging in sizes, parameters and dates.

IV. Results

Our experiments evaluated three algorithms—Greedy, Simulated Annealing, and Reinforcement Learning (Q-learning)—across different meal planning scenarios. We tested these algorithms on three problem types: MinimizeWasteProblem, CountExpiredItemsProblem, and ParametersProblem. Our key metrics were waste reduction, solution quality, and execution time.

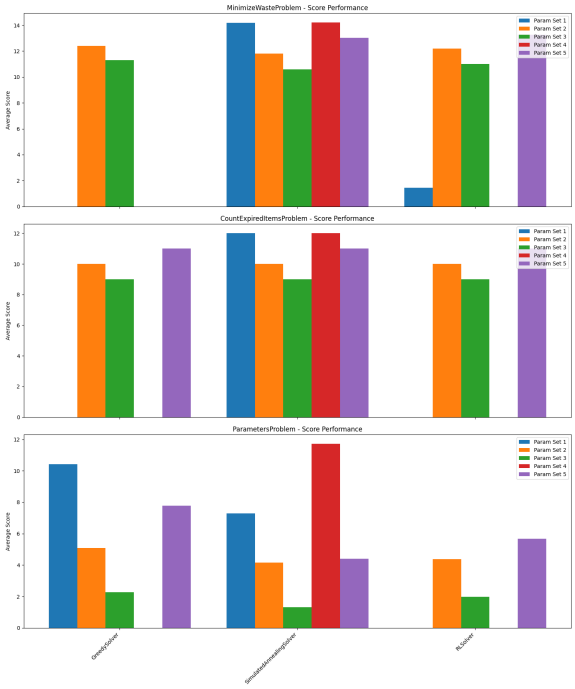

Fig. 1. Algorithm Score Overview

1 shows the overall score performance of each algorithm across all problem types. Simulated Annealing stands out as the top performer, consistently achieving the highest scores across various scenarios.

2 presents the execution time comparison. The Greedy algorithm is the fastest, followed closely by Simulated Annealing. Reinforcement Learning shows significantly longer run times.

A. Algorithm Analysis

1) Greedy Algorithm Performance: The Greedy algorithm excelled in speed but often failed to produce complete meal plans. Its strategy of making locally optimal choices led to situations where it ran out of ingredients before completing the required number of meals.

As seen in2, the Greedy algorithm consistently had the shortest execution times across all problem types. However, its speed came at a cost to solution quality.

2) Simulated Annealing Performance: Simulated Annealing emerged as the most effective algorithm in our tests. It achieved the highest scores in most scenarios and solved almost every meal planning problem we presented.

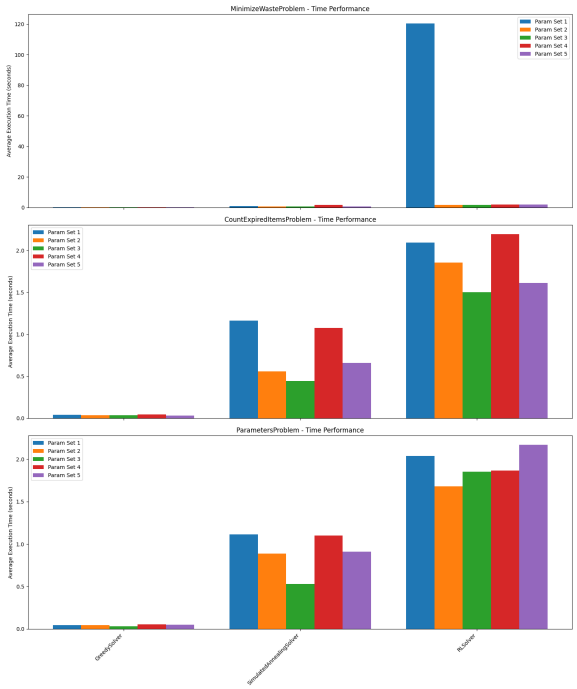

Fig. 2. Algorithm Runtime Overview

1 illustrates Simulated Annealing's superior performance in terms of solution quality. It consistently outscored the other algorithms across all problem types.

Simulated Annealing showed particular strength in the MinimizeWasteProblem. We believe this is due to the nature of the problem's score function. The MinimizeWasteProblem likely has a smoother, more convex score landscape, allowing Simulated Annealing to behave similarly to gradient descent.

3) Reinforcement Learning Performance: The Reinforcement Learning (Q-learning) algorithm showed potential but faced challenges. Its scores were comparable to Simulated Annealing in many cases, demonstrating its ability to find good solutions.

However, Reinforcement Learning had two significant drawbacks:

1) Long execution times: As seen in 2, Reinforcement Learning consistently took much longer to run than the other algorithms.
2) Sensitivity to hyperparameters: The algorithm's performance varied greatly depending on its hyperparameter settings.

To address the hyperparameter challenge, we conducted a grid search to find optimal settings. The following table shows the top-performing configurations for MinimizeWasteProblem and CountExpiredItemsProblem, respectively.

This I shows that the best hyperparameter settings differ between problem types. This highlights the challenge of using Reinforcement Learning in practice, as it requires tuning for each specific problem.

TABLE I
TOP 5 HYPERPARAMETER CONFIGURATIONS FOR BOTH MINIMIZEWASTEPROBLEM AND COUNTEXPIREDITEMSPROBLEM

Learning Rate	Discount Factor	Epsilon	Episodes	MinimizeWaste	CountExpired
0.001	0.99	0.2	100	**17.621**	11.0
0.1	0.95	0.2	100	16.647	11.0
0.001	0.9	0.3	1000	16.642	11.0
0.01	0.9	0.2	500	16.604	11.0
0.001	0.9	0.3	100	16.604	11.0
0.001	0.95	0.2	1000	16.604	11.0
0.1	0.99	0.2	100	16.604	11.0
0.01	0.95	0.2	1000	16.604	11.0
0.01	0.95	0.3	500	16.597	11.0
0.01	0.9	0.3	500	16.537	11.0
0.1	0.9	0.3	500	13.270	**14.0**
0.01	0.95	0.2	500	15.460	**14.0**
0.1	0.9	0.1	100	13.270	13.0
0.1	0.9	0.1	1000	13.270	13.0
0.1	0.9	0.2	100	15.460	13.0
0.1	0.9	0.2	1000	13.270	13.0
0.1	0.99	0.1	100	13.270	13.0
0.1	0.99	0.3	100	15.490	13.0
0.01	0.9	0.1	1000	13.270	13.0
0.01	0.9	0.3	100	16.307	13.0

Bold is highest, underline is second highest.

B. Performance Across Problem Types and Problem Transfer

Each problem type—MinimizeWasteProblem, CountExpiredItemsProblem, and ParametersProblem—produced different results with each algorithm.

In 3 we can see the results of each of the algorithms on different datasets with different parameters (Start time, number of meals etc.). Simulated Annealing clusters near the top-right corner, showing high and consistent performance across problem types. Reinforcement Learning shows more spread, indicating its performance varies more between problems, the algorithm showed more consistent performance across problem types compared to the Greedy algorithm, but still fell short of Simulated Annealing's results

The Greedy algorithm performs similar to Simulated Annealing on problem layouts it was able to solve, but we can see the number of solutions is sparse, meaning it wasn't able to find a solution for many of the problems.

We wanted to check the ability of learning a policy using RL for a heuristic function, so we tried to analyze the ability of the RL solver to learn one problem i.e MinimizeWasteProblem, and use the policy gathered for CountExpiredItemsProblem which is a more accurate depiction of our problem.

4 shows how well the algorithm transfers between different problem types. Points closer to the diagonal line indicate better transfer, meaning the algorithm performs similarly on both problems, and as we can see CountExpiredItemsProblem is affected by raising scores on MinimizeWasteProblem, the transfer isn't perfect for obvious reason like the discretization of the score, but we can see that an effect exists.[1]

C. Impact of Problem Parameters

The main factor affecting algorithm performance was whether a valid meal plan existed under the given constraints. When we increased the number of days or meals per day, we sometimes created scenarios where no valid solution existed.

Simulated Annealing proved most robust in these cases. It usually found a solution when one existed, and quickly identified when no solution was possible. The Greedy algorithm often failed in these more constrained scenarios, while Reinforcement Learning sometimes found solutions but took much longer to do so.

D. Unexpected Findings

The strong performance of Simulated Annealing was unexpected. Despite being a relatively simple algorithm, it outperformed the more complex Reinforcement Learning approach in both score and runtime across the number of iterations and datasets we tested.

This result challenges the assumption that more complex algorithms always yield better results. It suggests that for this particular problem space, the balance of exploration and exploitation in Simulated Annealing is particularly effective.

E. Trade-offs Between Execution Time and Solution Quality

Our results clearly show the trade-offs between execution time and solution quality:

1) The Greedy algorithm offers the fastest execution but often fails to find valid solutions, especially in more constrained scenarios.
2) Simulated Annealing provides an excellent balance, offering near-optimal solutions with execution times close to the Greedy algorithm.
3) Reinforcement Learning can find high-quality solutions but at the cost of much longer execution times and the need for careful hyperparameter tuning.

These trade-offs highlight the importance of choosing the right algorithm based on the specific needs of a meal planning scenario. In time-sensitive situations where a good (but not necessarily optimal) solution is needed quickly, Simulated Annealing appears to be the best choice. For scenarios where computation time is less constrained and finding the absolute best solution is critical, Reinforcement Learning might be worth considering, provided sufficient time is allocated for hyperparameter tuning.

F. Practical Implications

These results have several practical implications for meal planning systems:

1) Simulated Annealing emerges as a strong default choice for most meal planning scenarios. Its consistent performance and reasonable execution times make it suitable for a wide range of applications.
2) The Greedy algorithm, while fast, should be used cautiously. It might be suitable for very simple scenarios or as a quick first pass to generate a baseline solution.
3) Reinforcement Learning, while powerful, requires careful consideration before deployment. Its long execution times and sensitivity to hyperparameters make it more

[1]This isn't a rigorous statistical observation, and the size of the effect is indeed minimal, yet in our opinion worth mentioning.

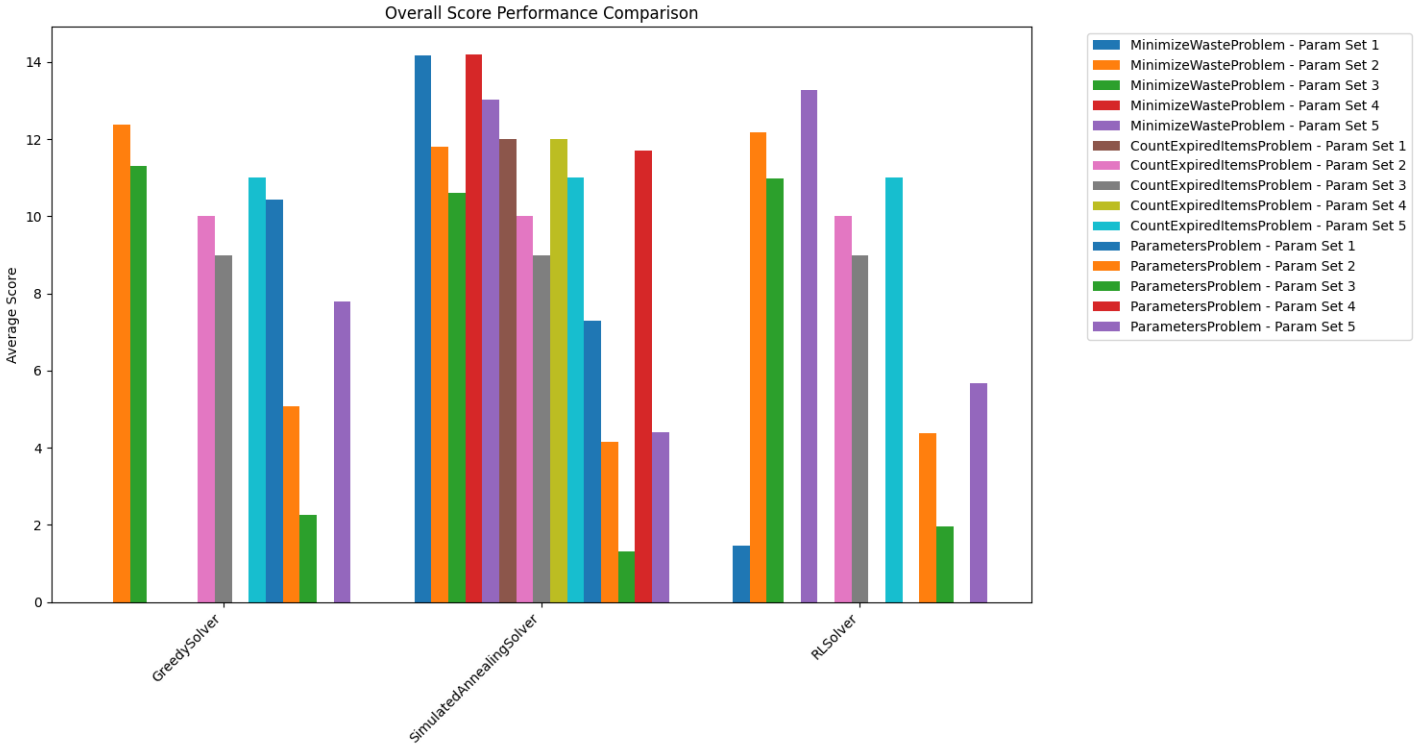

Fig. 3. Algorithms score on a variety of problem configurations

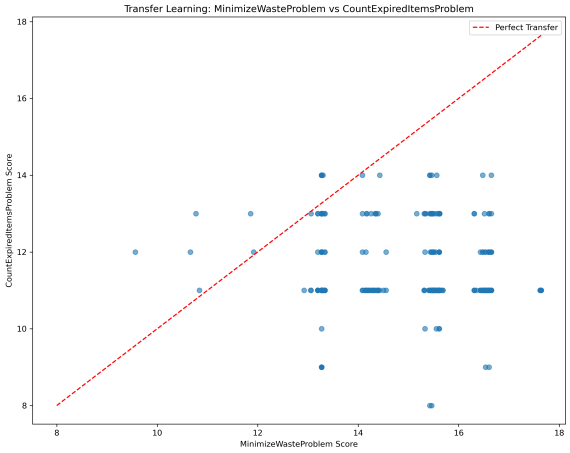

Fig. 4. Problem Transfer

suitable for offline optimization or scenarios where extensive computation resources are available.

4) The choice of algorithm should consider the specific problem type. For example, Simulated Annealing's particular short runtime suggests it would be better for very large datasets.
5) Robust meal planning systems should incorporate multiple algorithms. Using Simulated Annealing as a primary solver with a RLHF algorithm for creating personalized recipes and meal plans would make for a great platform.

V. Summary

Our project tackled the challenge of optimizing food inventory management in kitchens using AI-driven planning techniques. We aimed to minimize food waste while considering factors like personal taste, preparation time, and shelf life. We modeled the problem as a state space search. Each state represented the current inventory and meal plan, while actions were recipe selections that transformed the state by using ingredients and adding meals to the plan. Our goal was to find a sequence of actions that optimized meal planning based on our evaluation criteria.

We implemented and compared three algorithms: Greedy, Simulated Annealing, and Reinforcement Learning (Q-learning). Each algorithm approached the problem differently, offering unique trade-offs between solution quality and execution time.

Our results revealed several key findings:

1) Simulated Annealing emerged as the most effective algorithm overall. It consistently achieved the highest scores across various scenarios and problem types, while maintaining reasonable execution times.
2) The Greedy algorithm was the fastest but often failed to produce complete meal plans, especially in more constrained scenarios.
3) Reinforcement Learning showed potential in finding high-quality solutions but suffered from long execution times and high sensitivity to hyperparameter settings.
4) Performance varied across problem types (MinimizeWasteProblem, CountExpiredItemsProblem, and Param-

etersProblem), with Simulated Annealing showing particular strength in the MinimizeWasteProblem.
5) The main factor affecting algorithm performance was whether a valid meal plan existed under the given constraints.

These findings have practical implications for meal planning systems. Simulated Annealing stands out as a strong default choice for most scenarios, while the Greedy algorithm might serve as a quick baseline or fallback option. Reinforcement Learning, despite its potential, requires careful consideration due to its computational demands and sensitivity to hyperparameters.

Critiquing our work, we acknowledge several limitations and areas for improvement:

1) Assumptions: Our model assumes accurate data on ingredient expiration dates and nutritional values. In reality, this data might be imperfect or missing, which could significantly impact our results. Future work could explore robust optimization techniques to handle uncertain or incomplete data.
2) Real world applications: While we tested our algorithms on various dataset sizes, we didn't fully explore their performance on very large, real-world scale problems. This is crucial for understanding how these algorithms would perform in commercial or industrial settings. Our current model doesn't account for new items being added to the inventory over time, which is a common occurrence in real kitchens.
3) User preferences: Our model considers taste as a parameter but doesn't account for individual user preferences or dietary restrictions. Incorporating these factors could make the system more personalized and practical for real-world use.
4) Parameter optimization: While our system allows for optimization of nutritional parameters, we didn't deeply explore how well the generated meal plans meet comprehensive nutritional guidelines. This is an important aspect for any practical meal planning system. And although our system aims to reduce food waste, we didn't explicitly model or optimize for other environmental factors like greenhouse gas emissions associated with different ingredients or cooking methods.

If we were to revoke certain assumptions, such as the accuracy of expiration dates or the static nature of the inventory, our algorithms would need significant modifications. For example, we might need to incorporate probabilistic models for food spoilage or implement dynamic programming approaches to handle changing inventories.

Other methods that could potentially solve this problem include:

1) Genetic Algorithms: These could be useful for exploring a wider solution space and potentially finding novel meal combinations, these perform in similar manner to Simulated Annealing but allow for more complex policies to arise.
2) Mixed Integer Programming: This could provide optimal solutions for smaller problem instances and serve as a benchmark for our heuristic approaches, in fact this is the main method used today, as seen in previous work section.
3) Deep Reinforcement Learning: More advanced RL techniques might use an Actor-Critic architecture to both determine best values for each recipe based upon personal taste and chosen parameters, and optimize the meal plan for that specific reward function.

In conclusion, our work provides a solid foundation for AI-driven meal planning systems aimed at reducing food waste. While Simulated Annealing emerged as the most effective approach in our tests, there's ample room for further research and refinement. Future work should focus on addressing the limitations we've identified and exploring additional optimization techniques to create more robust, flexible, and practical meal planning solutions.

REFERENCES

[1] J. L. Buttriss, A. Briend, N. Darmon, E. L. Ferguson, M. Maillot, and A. Lluch. Diet modelling: How it can inform the development of dietary recommendations and public health policy. *Nutrition Bulletin*, 39(1):115–125, 2014.
[2] Jill Freyne and Shlomo Berkovsky. Intelligent food planning: personalized recipe recommendation. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, IUI '10, pages 321–324, New York, NY, USA, 2010. Association for Computing Machinery.
[3] M.A. van Rooijen, J.C. Gerdessen, G.D.H. Claassen, and S.L.J.M. de Leeuw. Optimizing household food waste: The impact of meal planning, package sizes, and performance indicators. *Resources, Conservation and Recycling*, 205:107559, 2024.