PITTGRUB: A FRUSTRATION-FREE SYSTEM TO REDUCE FOOD WASTE BY NOTIFYING HUNGRY COLLEGE STUDENTS

by

Mark Silvis

B.S. in Computer Science, University of Pittsburgh, 2015

Submitted to the Graduate Faculty of the

Kenneth P. Dietrich School of Arts & Sciences in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2018

PITTGRUB: A FRUSTRATION-FREE SYSTEM TO REDUCE FOOD WASTE BY NOTIFYING HUNGRY COLLEGE STUDENTS

Mark Silvis, M.S.

University of Pittsburgh, 2018

The amount of food waste generated by the U.S. is staggering, both expensive in economic cost and environmental side effects. Surplus food, which could be used to feed people facing food insecurity, is instead discarded and sent to landfills. Institutions, universities, and non-profits have noticed this issue and are beginning to take action to reduce surplus food waste, typically by redirecting it to food banks and other organizations or having students transport or eat the food. These approaches present challenges such as transportation, volunteer availability, and lack of prioritization of those in need. In this thesis, we introduce PittGrub, a notification system to intelligently select users to invite to events that have leftover food. PittGrub was invented to help reduce food waste at the University of Pittsburgh. We use reinforcement learning to determine *how many* notifications to send out and *whom* to prioritize in the notifications. Our goal is to produce a system that prioritizes feeding students in need while simultaneously eliminating food waste and maintaining a fair distribution of notifications. As far as we are aware, PittGrub is unique in its approach to eliminating surplus food waste while striving for social good. We compare our reinforcement learning approach to multiple baselines on simulated datasets to demonstrate effectiveness. Experimental results comparing various algorithms show promise in eliminating food waste while helping those facing food insecurity and treating users fairly. At the time of writing, our prototype is in beta; we plan to have it deployed during the Spring semester of 2018.

Keywords: reinforcement learning, machine learning, food insecurity, sustainability.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

PREFACE

This thesis is dedicated to my girlfriend, Emma. Thank you for your patience, encouragement, and never-ending support.
I would like to take this opportunity to thank all of those who have provided guidance and support. In particular, I would like to thank Alex Labrinidis for providing me with the opportunity to work with him and for navigating this thesis in a successful direction. I would also like to thank Anthony Sicilia for his momentous assistance with completing experiments and plotting the results. I am grateful to the committee for their time and valuable feedback. Lastly, would like to thank my friends, fellow students, and parents for their moral support.

1.0 INTRODUCTION

Every year, nearly 400 billion pounds of food is circulated through the U.S. food supply chain. It travels from farms to distribution centers to retailers until finally food service managers and grocery stores supply our institutions and homes. Much of this food, however, never makes it to the plate. Approximately 40%, or 160 billion pounds, of this food is left uneaten, sent to landfills where it makes up 21% of held waste, the largest contributor [5]. Wasting such a large portion of our food is an enormous resource sink — we exhaust 21% of U.S. fresh water, 19% of all fertilizer, and 18% of all U.S. cropland on the waste alone, totaling $218 billion (1.3% of the GDP) [18]. Reducing the amount of food waste is just as much a humanitarian problem as it is an economic one. Reclaiming the wasted food could feed all of the 42 million Americans facing food insecurity three times over [5].

Of the various stages of the food supply chain, restaurants and food service institutions generate a whopping 26% of waste, the second highest behind households [18]. Oftentimes, this waste consists of scraps that, although unfit for human consumption, can be used for animal feed or composted rather than sent to a landfill; however, a large portion of wasted food is edible, yet simply discarded. A 2016 audit by Sodexo Dining Services at the University of Pittsburgh discovered that of the 338.8 pounds of waste generated daily at their dining hall “The Perch”, 92 pounds was recoverable surplus food [8]. There are four more dining halls just like that one on Pitt’s campus. This is food that could be repurposed to feed those in need, one of the EPA’s top recommend actions to reduce food waste [1].

Fortunately, there are many organizations determined to recover the surplus and help feed the hungry. 412 Food Rescue is a Pittsburgh-based operation that recovers surplus food from institutions and redirects it to local non-profits that benefit from the donation but do not have the resources to recover the food themselves [19]. Food Recovery Network (FRN) is

Table 1: A summary of features provided by PittGrub compared to other surplus food waste recovery services: Titan Bites, 412 Food Rescue, and Food Recovery Network.

	PittGrub	Titan Bites	412 Food Rescue	FRN
Reduce food waste	✓	✓	✓	✓
Prioritize food-insecure students	✓		✓	✓
Impromptu leftovers	✓	✓	✓	
No transportation required	✓	✓		
No storage required	✓	✓		
Notifies consumers	✓	✓		
Notifications limited by food volume	✓			

another non-profit which organizes student volunteers to collect and transport surplus food from college dining halls to food banks [14]. Collecting food surplus in such a way comes with a set of challenges. It requires active volunteers with adequate transportation, safely storing the food before and during transportation, and a food bank or organization willing to accept the food. Additionally, these food pickups are usually organized ahead-of-time, thus making it more difficult to recover food impromptu, such as after a catered event or conference.

Another approach to surplus food recovery is to have the people come to the food. Titan Bites, an app developed by Auxiliary Services Corp.'s Campus Dining Services, allows students at California State University at Fullerton to opt-in to notifications of leftover food events so that they can finish the surplus. In addition to reducing waste, Titan Bites aims to provide students facing food insecurity a simple way to receive free meals [25]. However, Titan Bites notifies their entire user base of leftover food events, with no way of prioritizing students currently interested in free food or those in need. Additionally, notifications are not limited by the number of leftover servings, possibly causing more students to show up than there is food.

To address the challenges faced by other services recovering surplus food, we developed

PittGrub, a mobile application that intelligently notifies students when there are catered events at the University of Pittsburgh with leftover food. We use reinforcement learning techniques to adjust the number of notifications sent out relative to the number of leftover servings and our user-base's probability of attending. In addition, we prioritize sending out notifications to students in need while also considering fairness of our selection, so that a significant portion of our users receive a notification over a large enough number of events.

The key contributions of this thesis are as follows:

- **Developed PittGrub, a practical system of reducing food waste:** We developed a practical system to reduce surplus food waste at the University of Pittsburgh by notifying students to attend events with leftover food. We avoid the problems of transportation and volunteer availability by having hungry students travel to the location of the food instead of vice-versa. Additionally, we promote the humanitarian issues of food waste by prioritizing students in need, while also aiming for a fair notification distribution across the user population. Although our app has a name specific to our university, the solution is trivially generalizable to other institutions.
- **Solved two difficult problems of managing leftover food after events:** PittGrub solves two difficult problems with managing events with leftover food: *whom* to invite and *how many* to invite. PittGrub uses reinforcement learning to determine a *booking factor*, which is an adjustment to the number of leftover servings to attain a precise level of attendance; ideally, the booking factor avoids wasting food caused by underbooking, while also avoiding an understock scenario caused by overbooking (which would be frustrating to the users). Additionally, PittGrub values users based on three factors: 1) their likelihood of attending an arbitrary event, 2) whether they face food insecurity, and 3) how fair it is to invite them (i.e., they have not been invited for a while).
- **Defined metrics for evaluating notification selection performance:** We defined three metrics for evalutating our selection of notification candidates that ensures quality candidate selection while assisting those facing food insecurity.

- **Experimentally evaluated our proposed system against multiple baselines:** We compared our proposed method to multiple baseline approaches that maximize a subset of the metrics defined. These experiments determine the effectiveness of the approaches relative to their trade-offs.

1.1 ORGANIZATION

This thesis is organized as follows:

Chapter 2 describes the current functionality of the PittGrub application. It includes the technologies used and the tasks which our team focused on during the initial deployment phase. Chapter 3 includes the necessary background information for the thesis, including how we define food insecurity and the defined metrics and baselines. In Chapter 4, I describe our approach to selecting the set of users to notify for an event with leftover food. I break this problem into two problems: *whom to invite* and *how many to invite*. To choose *whom to invite*, we reframe the problem of user selection as the 0-1 knapsack problem and evaluate users by social criteria. To determine *how many to invite*, we adjust the number of servings of leftover food by a multiplicative *booking factor* to minimize waste while avoiding an understock scenario. Chapter 5 includes the experiments used to evaluate our proposed method against multiple baseline approaches. In Chapter 6, I review our work and key contributions. Finally, in Chapter 7, I outline the future work that we have planned and some preliminary results.

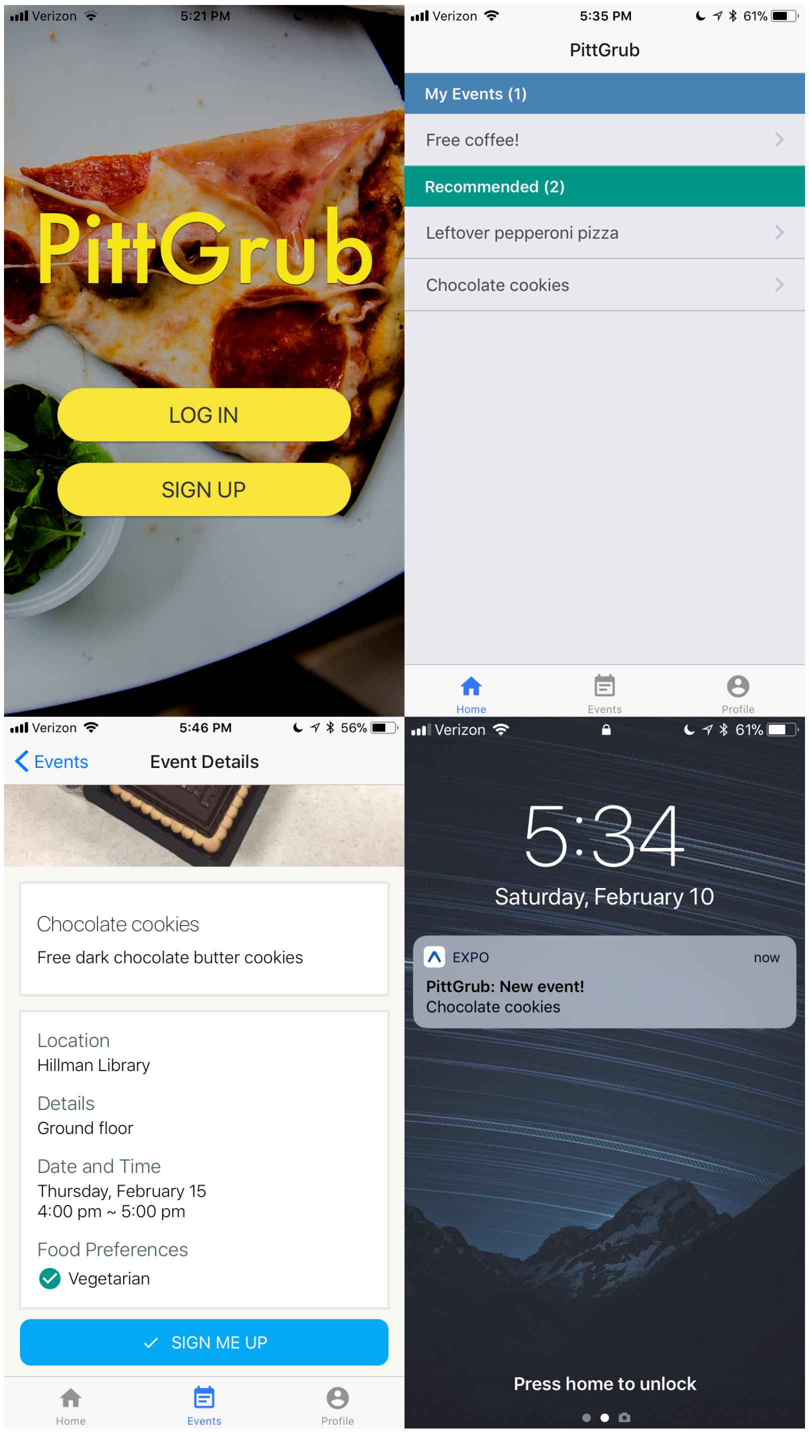
2.0 PROTOTYPE

At the time of writing this thesis, we have a working prototype of PittGrub available to students to download on iOS and (soon) Android [2]. Pitt students can create an account using their Pitt email address and set their food preferences (gluten-free, dairy-free, vegetarian, and vegan). Certain users called "hosts", approved by us, can post events with details like servings, date and time, location, food preferences, and a photograph. All users whose food preferences match the characteristics of a given event receive a notification once it is posted; for example, vegetarians will not receive notifications for events that are not categorized as vegetarian-friendly, but non-vegetarians, barring other restrictions, will be notified.[1] Figure 1 comprises screenshots of most of the current functionality. In addition to the mobile application, we have a website, as shown in Figure 2, which provides account management functionality, news, instructions, and frequently asked questions. The website is located at `https://pittgrub.com`.

The PittGrub server runs on an Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instance [4]. It was built with Python 3 using the Tornado server framework [27]. SQLAlchemy [26] is used to connect to the MySQL database, located on an AWS Relational Database Service (RDS) instance [23]. AWS Elastic Beanstalk is used to serve the website [3]. AWS Simple Email Service (SES) is used to send email alerts [24]. See Figure 3 for an overview of our technology stack.

The PittGrub mobile application was written using the React Native [13] framework for fast prototyping. It is currently distributed via the developer tool Expo [6], which provides installation of React Native-based apps. As we are still in beta testing, distribution via Expo

[1]Note that the sophisticated notification techniques developed by this thesis are not yet in PittGrub. Incorporating them is one of our immediate next steps.



PittGrub

My Events (1)

Free coffee!

Recommended (2)

Leftover pepperoni pizza

Chocolate cookies

Event Details

Chocolate cookies

Free dark chocolate butter cookies

Location
Hillman Library

Details
Ground floor

Date and Time
Thursday, February 15
4:00 pm ~ 5:00 pm

Food Preferences
✓ Vegetarian

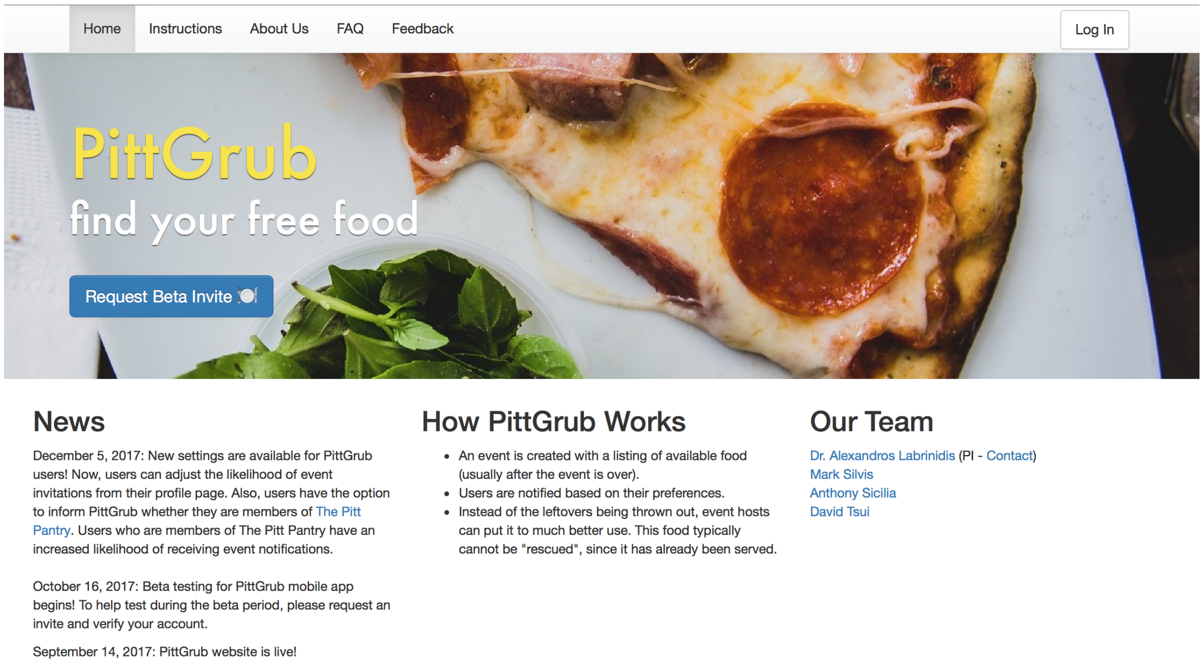
5:34

Saturday, February 10

EXPO now

PittGrub: New event!
Chocolate cookies

Figure 1: An overview of the current functionality of the PittGrub app. A user can log-in and view events, and approved users can post events. Users whose food preferences match those of a posted event will receive a notification (e.g., vegetarians will never receive notifications for non-vegetarian-friendly events).



News

December 5, 2017: New settings are available for PittGrub users! Now, users can adjust the likelihood of event invitations from their profile page. Also, users have the option to inform PittGrub whether they are members of The Pitt Pantry. Users who are members of The Pitt Pantry have an increased likelihood of receiving event notifications.

October 16, 2017: Beta testing for PittGrub mobile app begins! To help test during the beta period, please request an invite and verify your account.

September 14, 2017: PittGrub website is live!

How PittGrub Works

- An event is created with a listing of available food (usually after the event is over).
- Users are notified based on their preferences.
- Instead of the leftovers being thrown out, event hosts can put it to much better use. This food typically cannot be "rescued", since it has already been served.

Our Team

Dr. Alexandros Labrinidis (PI - Contact)
Mark Silvis
Anthony Sicilia
David Tsui

Figure 2: A screenshot of the PittGrub website. The website focuses on providing account management, news, instructions, and answers to frequently asked questions.

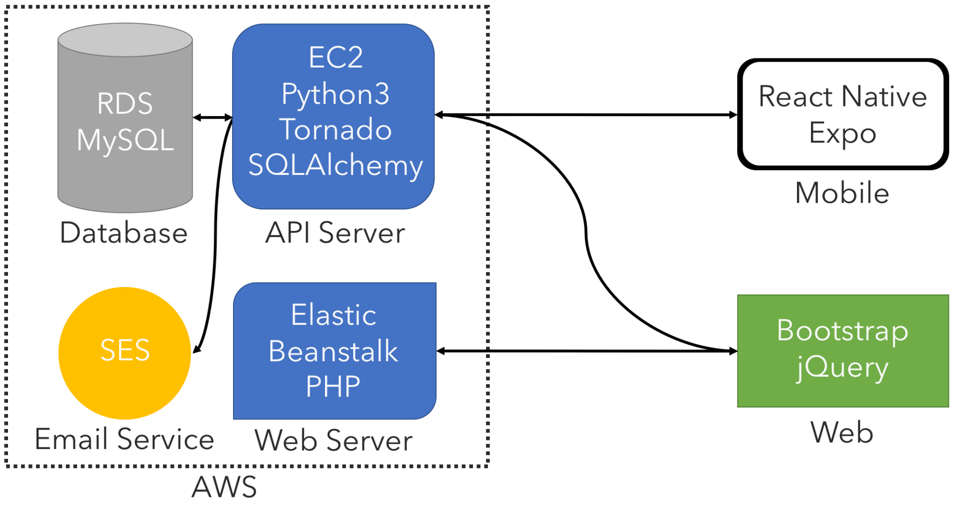


Figure 3: PittGrub technology stack.

provides a fast and convenient way to send app updates to users while avoiding the iOS App Store requirements. At the time of writing, we have nearly 200 users signed up for the beta.

2.1 NEXT STEPS

The next steps for the mobile app include:

1. Incorporating the methods from this thesis into production
2. Including functionality for hosts to alert interested users when the food is all out
3. Making PittGrub available on both the iOS App Store and Google Play Store
4. Aggressively promoting PittGrub in order to gain traction

To spread awareness of our app, we have had multiple discussions with University Dining Services, the Student Office of Sustainability, and numerous student groups. We have established a website for PittGrub [16] that includes news and installation instructions.

2.2 APPLICATION WORKFLOW

Figure 4 provides an overview of the full PittGrub application workflow. First, a host (i.e., a user with event posting privileges) creates an event with leftover food using the PittGrub mobile app (Figure 4 step 1). Second, the server receives the event information and updates the database, simultaneously querying for the current set of users and their parameters (Figure 4 step 2). Next, given the event and user parameters, the server receives the set of users to notify from the notification system (Figure 4 step 3). The notification system could be as simple as a basic filter, or as complex as our outlined method in Chapter 4. Then, the server invites the users as provided by the notification system, ignoring the others (Figure 4 step 4). In the case of Figure 4, Users 1 and 2 are invited while User 3 is not. Perhaps User 3 is a vegetarian, and the event did not cater to their preferences. Finally, User 1 decides that they are interested in the event, so they let PittGrub know this by clicking the *Sign Me Up* button on the event page (Figure 4 step 5).

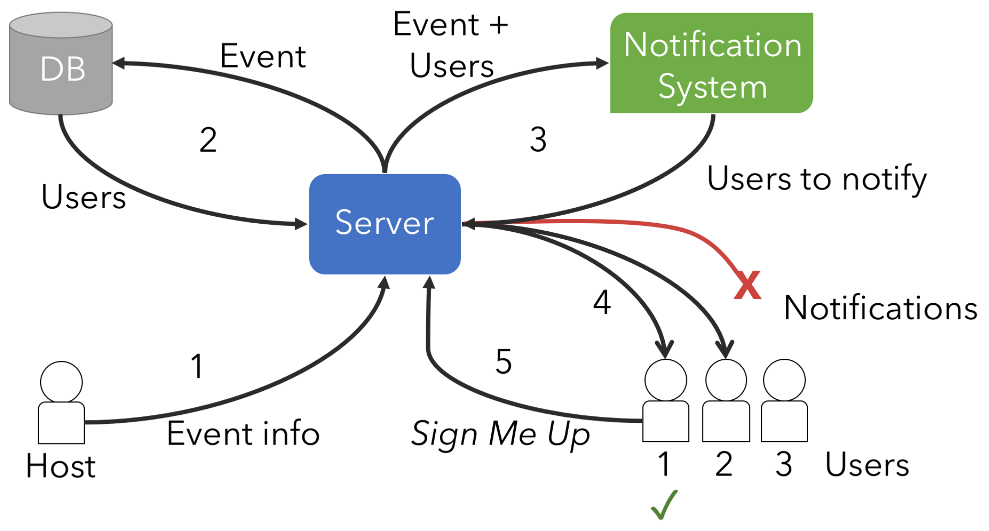


Figure 4: Workflow of the PittGrub application: **1)** the host creates an event with leftover food; **2)** the server updates the database with the new event and fetches the current set of users; **3)** the notification system provides the server with the set of users to notify given the event and user parameters; **4)** the server sends the event invitation to users 1 and 2, but not user 3; **5)** user 1 is interested, so they click the *Sign Me Up* button.

3.0 BACKGROUND

For the experimental evaluation of this thesis, our codebase was written in Python 3, using NumPy [15] for most of our computation and Scikit-learn [21] for other convenience methods. Seaborn [22] and Matplotlib [12] were both essential in generating plots to visualize and learn from the data.

3.1 THE PITT PANTRY

The National Student Campaign Against Hunger & Homelessness defines food insecurity as "the lack of reliable access to sufficient quantities of affordable, nutritious food". According to their 2016 report on the challenge of food insecurity for college students, approximately 48% of respondents reported experiencing food insecurity in the past 30 days [9].

The University of Pittsburgh has a program through the Student Office of Sustainability called the Pitt Pantry which helps to ensure that food-insecure students have regular access to healthy food options [17]. The Pitt Pantry allows members to collect donated food in a style similar to food banks. In 2016, the Pitt Pantry had more than 300 students take advantage of their services [7]. Given that this is only 1% of the Pitt student population, there is clearly a large discrepancy between the number of college students facing hunger and the number utilizing the Pitt Pantry's services. The PittGrub team aims to promote the Pitt Pantry's services by prioritizing users who voluntarily and privately inform PittGrub that they are a member of the program. By prioritizing these students, we are striving to improve their access to healthy food choices.

Given our goal of assisting students facing food insecurity, much of this thesis is focused on our approach to prioritizing these students, in particular, students who are members of the Pitt Pantry. For the remainder of this thesis, we will often refer to students facing food insecurity as members of the Pitt Pantry, or Pantry students for short. We assume that PittGrub users will self-identify as Pitt Pantry students through the app; this information is kept private as part of a user's application profile.

3.2 METRICS

The purpose of our work is to develop a user selection algorithm that determines which users should be notified of leftover food. The primary property we wish to achieve is the minimization of food waste. In addition, we define three user-specific qualities that an ideal selection algorithm would be able to optimize:

1. Frustration caused by too many unwanted notifications (minimize)
2. Notification distribution fairness over a number of events (maximize)
3. Percentage of Pantry students notified per event (maximize)

To evaluate the performance of a user selection algorithm on each of the ideal qualities, we define the following metrics:

1. **Waste:** $d_e = S_e - w_e$ where S_e is the number of servings for event e and w_e is the number of notified users that showed up to event e
   a. **Understock:** defined as negative waste, i.e., we invite more users than there are servings
2. **Percent Went:** $\frac{w_e}{n_e}$ where w_e is the number of notified users who showed up to event e and n_e is the number of users notified for event e; this is our measurement for *frustration*
3. **Total Fairness:** $1 - \sum_{i=1}^{n} \frac{M - x_i}{M * n}$ where x_i is the number of notifications received by user i over some subset of events E and M is the maximum number of notifications received by any user over this subset of events
4. **Percent Pantry:** $\frac{\sum_{i=1}^{n_e} y_{i,e}}{n_e}$ where n_e is the total number of users notified for event e and $y_{i,e} \in \{0, 1\}$ is 1 for user i notified of event e if i is Pantry and 0 otherwise

Note that each metric which is specific to some event e in a subset of events E can be generalized to an aggregate metric over the subset of events E by simply averaging. For our later experimentation, we focus on the use of aggregate metrics.

3.3 BASELINES

Each of the previously described user metrics can be maximized using a specific baseline algorithm. The term capacity is used to refer to the number of leftover servings from an event. The baselines are as follows:

1. **Frequent-First**: select the users most probable to show up in-order until reaching capacity. This algorithm is expected to maximize *Percent Went*, i.e., minimize *frustration*.
2. **Round-Robin**: select users in a queued fashion, never notifying a user again prior to all others receiving a notification, until reaching capacity; performed over multiple events. This algorithm is expected to maximize *Total Fairness*.
3. **Pantry-First**: select the Pantry students first, followed by non-Pantry students arbitrarily, until reaching capacity. This algorithm is expected to maximize *Percent Pantry*.

The baseline algorithms are used in the Experiments chapter for comparison to our proposed method. There is no baseline for wasted food since it is an open optimization problem.

4.0 PROPOSED METHOD

In this section, we introduce our approach to solving both of the problems in managing leftovers: *whom* to invite and *how many* to invite. In the ideal scenario, we would maximize the portion of notifications sent to food-insecure students while maintaining a fair distribution, avoid creating an understock situation caused by inviting too many users, and never waste food. However, improving some metrics results in trade-offs with respect to the others; therefore, we must make compromises in our parameters and improvements.

4.1 WHOM TO INVITE

To solve the *whom to invite* problem, we assume that we have a black box probability estimator that generates a *probability score* per user. This estimator could take as input various user parameters, such as distance from the event and current user availability, to produce some likelihood of attending a given event. The probability score vector with components as user probabilities could be calculated as:

$$\mathbf{p} = P(\mathbf{Z})$$

where $\mathbf{Z}$ is a matrix with rows as users and columns as arbitrary user/event properties, and P is our black box estimator. Through the remainder of this thesis, we assume that we are provided a vector of probability scores comprising the probability for each user to attend an event.

A user's probability of attendance, while crucial in estimating the total number of users to invite, does not capture the social value of inviting said user: namely, whether they are food-insecure or if it is fair to invite them. To select users by their social value, in addition to probability of attending an event, we introduce our method of valuing users: Fair (food)Insecure Probability Score value model, or FIPS. FIPS evaluates users by their social value in addition to their general probability score. The FIPS model accepts a user's probability score, fairness score, and pantry value (0 or 1), along with a set of weights to scale the magnitude of each input, and produces a value for said user. The FIPS evaluation procedure can be calculated as:

$$\mathbf{v} = V(\mathbf{U}) = \mathbf{U}\mathbf{w}$$

where $\mathbf{U}$ is a matrix with rows $u_i = (p_i, f_i, y_i)$ for each user; here p_i represents the aforementioned probability score, f_i represents a user's fairness score, and y_i their Pantry status. A vector of weights, $\mathbf{w}$, is used to scale the influence per property. This matrix multiplication yields $\mathbf{v}$ a column vector of user values.

We remark that f_i, a user's fairness score, is a value between 0 and 1 that represents how fair it is to send them a notification. This is generated by receiving a binary string of 1's and 0's defining a user's notification *history* for an equal length set of events with the most recent event on the left. Using the history string as h and number of events n, we calculate a user's fairness score as:

$$f_i = 1 - \frac{h}{2^n - 1}$$

which produces a fairness score that decreases as a user is invited to more recent events and increases as they are omitted. Since we are aiming for a fair distribution, we update every user's notification history per event; users receiving a notification have a 1 prepended to their history, and all others receive a 0. The history is then pruned to the 10 most recent events.

FIPS produces a linear value for each user given said user's properties. This allows us to select a set of users for any arbitrary event using their probability of attendance, food insecurity status, and fairness. Additionally, by providing a set of weights, we can choose to scale some properties to have a greater effect on the generated value than the others. The generalized model, which we call FIPS+, can generate values for users with arbitrary properties.

The next challenge is selecting a set of users for an event using the user values as the selection criteria, which we do by framing the problem as the knapsack problem.

4.1.1 The Knapsack Problem

The knapsack problem is a classic combinatorial optimization problem of attempting to select a subset of items, each with a value and a weight, such that the total value of the subset is maximized while constraining the total weight to some capacity. In our approach, we treat our user population as the set of items to select from, using their probability scores as the item weights and FIPS values as the item value. The subset is selected such that the sum of the subset weights reaches some capacity, which is set to the number of servings for the event in question. This version of the knapsack problem is known as the 0-1 knapsack problem, as we can only select 0 or 1 of each item, i.e., whether or not to notify the user. For a set of n users, with each user i having a probability score of p_i and FIPS value of v_i, each being selected ($x_i = 1$) or not selected ($x_i = 0$), and an event with S servings, we want to:

$$\max \quad \mathbf{x}^{\mathrm{T}}\mathbf{v} \qquad \text{s.t:} \quad \mathbf{x}^{\mathrm{T}}\mathbf{p} \leq S$$

$$\text{where} \quad \mathbf{v} = (v_i)_{i=1}^{n}, \quad \mathbf{x} = (x_i)_{i=1}^{n}, \quad \text{and} \quad \mathbf{p} = (p_i)_{i=1}^{n} \qquad (i).$$

The optimal solution to the 0-1 knapsack problem is a pseudo-polynomial time dynamic programming algorithm to select the item subset. This algorithm takes $\mathcal{O}(nS10^d)$ time, where S is the number of servings (i.e., capacity), n is the number of weights (i.e., user population), and d is our decimal precision. This approach is impractical for our purposes, as, on a modest machine, it takes approximately 5 minutes to select from 1000 users for a single event with 60 extra servings. Extrapolating further, notifying a 1000 user population of 100 events of various serving sizes would take over 8 hours of computation time. Given the perishable nature of the food for which we are inviting people, such long computation times are clearly unreasonable.

An alternative approach to solving the knapsack problem is to greedily select from the user population sorted by descending value until reaching capacity, which runs in $\mathcal{O}(n)$ time. On the same machine, this takes just over 1 minute to notify our 1000 user population of

100 events with varying serving sizes. The tradeoff with this algorithm is that it is sub-optimal, only guaranteeing that the total value of the subset is at least one-half the optimal [11]. However, given that we are developing this system for a real application, limitations in computation require us to use the greedy knapsack algorithm. One thing to note is that, since we care about maintaining user fairness, we must re-compute their fairness score and regenerate every user's FIPS value for every event selection decision. Since this requires re-sorting the users by their values for greedy selection, our greedy knapsack algorithm is actually $\mathcal{O}(n \log n)$. We remark that the introduction of the knapsack problem allows us to split our baseline algorithms into two variants, one that counts users as a single serving, and one that uses the probabilistic weight constraint of the knapsack problem, seen in (i).

4.2 HOW MANY TO INVITE

Now that we can prioritize users by their value, we must consider the robustness of our constraint in equation (i). In some cases, it is possible that our black box probability estimator P has some directional error or bias – it may, in general, over- or under-estimate the probability scores of our user population – we account for this possibility via a booking factor which we call ω. This modifies the 0-1 knapsack problem (i) as below

$$\max \quad \mathbf{x}^{\mathrm{T}}\mathbf{v} \qquad \text{s.t:} \quad \mathbf{x}^{\mathrm{T}}\mathbf{p} \leq \omega S \qquad (ii).$$

Manual selection of ω is not necessarily conceivable because we have no way of discerning the degree and direction of any potential bias our probability estimator may have. Our solution is to use Q-learning, which is a commonly used reinforcement learning algorithm, to determine ω.

4.2.1 Q-Learning

Inspired by behaviorist psychology, reinforcement learning consists of an agent that explores some state space by taking actions and receiving rewards or punishments. In this way, the agent may learn the correct actions to take given a particular state by exploring which ones generate the best reward long-term. It is an unsupervised technique, which means the optimal solution, if known, is not provided [20].

Q-learning, as a variant, is a model-free approach which learns an action-value function, Q, mapping state-action pairs to values; the agent can then decide on some optimal policy for action selection in a given state by choosing the action with the highest value. In practice, this decision comes with some level of trade-off between exploration and exploitation as the agent learns [20, 10].

Our proposed Q-learning algorithm is **booQ** (pronounced *book*). At a high level, the algorithm updates the booking constant ω over time, learning from the aforementioned waste (or understock) metric. The current range of observed average waste is considered to be our agent's state space. We discretize the state space by normalizing an observed waste d through a map:

$$d \mapsto d' \in [0, 1].$$

The normalized space is then uniformly discretized via a map:

$$d' \mapsto s \in \{1, 2, ..., n\}$$

where n is a previously specified number of intervals, which must be odd because the computation of the median interval (as an integer) is a key component to reward calculation – after normalization, the median interval corresponds to average waste being as discernably close to zero as possible. With regards to discernability, the number of intervals determines the resolution of booQ after discretization; more intervals correspond to a higher resolution. Through composition of the above, we have a positive integer representation of an observed average waste

$$d \mapsto s \in \{1, 2, ..., n\} \qquad (iii)$$

The accompanying action space consists simply of incrementing, decrementing, or leaving the booking constant, ω, unchanged. Respectively, we identify these actions $a \in \{-1, 1, 0\}$.

Algorithm 1 booQ

```
procedure booQ:
  inputs:
    $n$: odd number of intervals        $\tau$: decreasing exploration rate
    $\alpha$: decreasing learning rate   $\varepsilon$: exploitation constant $\in [0,1]$
    $\gamma$: discount factor            $\eta_{\max}$: a maximum step size
  initialize:
    $\omega_0 \leftarrow 1$
    $Q(s,a) \leftarrow 0$ for all $(s,a) \in \{1,2,...,n\} \times \{-1,0,1\}$
    select $a_0 \in \{-1,0,1\}$ randomly
    observe all other initial states without updating $Q$
  repeat with timestep $t$:
    perform notifications with booking constant $\omega_t$
    observe waste $d_t$
    $M \leftarrow \max\{|d_1|,|d_2|,...,|d_t|\}$
    $s_t \leftarrow \min\{\lfloor \frac{n(d_t+M)}{2M} \rfloor + 1, n\}$
    if $s_t \neq \frac{n+1}{2}$:
      $\rho_t \leftarrow -\left|\frac{n+1}{2} - s_t\right|$
    else:
      $\rho_t \leftarrow \frac{n+1}{2}$
    if $\rho_t \neq \rho_{t-1}$:
      $r \leftarrow \rho_t - \rho_{t-1}$
    else:
      $r \leftarrow \rho_t$
    $Q(s_{t-1},a_{t-1}) \leftarrow (1-\alpha)Q(s_{t-1},a_{t-1}) + \alpha(r + \gamma \max_a Q(s_t,a))$
    select $a_t$ by rule:
      $a_t \leftarrow \arg\max_a Q(s_t,a)$ with probability $1 - \max\{\varepsilon,\tau\}$
      otherwise:
        $a_t \leftarrow$ randomly from $\{-1,0,1\} - \{\arg\max_a Q(s_t,a)\}$
    $\eta \leftarrow \eta_{\max}\left(\frac{1}{1+e^{-|d_t|}}\right)$
    $\omega_{t+1} \leftarrow \omega_t + a_t\eta$
  end repeat
```

The magnitude of an increment or decrement is determined by the step size, η, which is computed via a function of the average waste over some series of events

$$\eta = \frac{\eta_{\max}}{1 + e^{-|d|}}$$

where $\eta_{\max}$ is a previously specified maximum step size; the use of a half-sigmoid allows for the step size to dynamically increase and decrease as average waste moves away from and closer to zero. The algorithm is thereby responsive to drastic changes in the user population and can quickly recover from incorrect decisions. As mentioned, the reward process relies heavily on the discretization process (*iii*). A rank, ρ, is assigned to each of $\{1, 2, ..., n\}$ based on distance from the median interval with closeness to the median corresponding to larger ρ

$$\rho = -\left|\frac{n+1}{2} - s\right|$$
$$\text{where } s \in \{1, 2, ..., n\} - \{\frac{n+1}{2}\}$$
$$\text{otherwise} \quad \rho = \frac{n+1}{2}.$$

The reward, r, is then computed by observing the difference between the current state's rank and the previous state's rank; here, rewards are given so that our agent is encouraged to take actions that result in a sequence of states which travel towards, and remain within, the median interval. As an example, if $\rho_t > \rho_{t-1}$, for t a time step, the reward r takes value

$$r = \rho_t - \rho_{t-1} > 0.$$

However, if $\rho_t < \rho_{t-1}$, then the reward r becomes a punishment with

$$r = \rho_t - \rho_{t-1} < 0.$$

If there is no change, the reward is exactly the current rank

$$r = \rho_t.$$

The algorithm uses a standard ε-decaying exploration/exploitation schedule as well as the standard Q-learning update rule [20, 10]. The specifics of booQ can be found in Algorithm 1.

Table 2: The rank of the intervals; i.e. states after discretization. The median interval (11) corresponds to near zero waste.

Discretized State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Rank	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	11	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10

During experimentation, we chose a discount factor of $\gamma = 0.1$ to value short-term rewards more highly than long-term rewards [10]; in practice, our agent should ignore long-term rewards as our student-based user population has a cyclic presence in the city – e.g. a booking constant over the summer term may need to be drastically larger than in the spring term. We chose $\eta_{\max} = 0.1$ to enforce a level of control over the degree which ω was altered at each iteration. We chose $n = 21$ to give our agent high resolution in determining the optimal booking constant; a lower number of intervals results in a wider range of accepted error within the booQ algorithm. Table 2 represents the rank-space of booQ with $n = 21$. We chose a decreasing learning rate [10], α, as a function of the number of times the current state-action pair was seen:

$$\alpha = \alpha\left(i_{s,a}\right) = \frac{1}{i_{s,a}}$$

where $i_{s,a}$ is the number of times booQ took a at s. We chose τ in our ε-decaying exploration/exploitation scheme as a function of the number of times the current state was seen:

$$\tau = \tau\left(i_s\right) = \frac{1}{\sqrt{i_s}}$$

where i_s is the number of times any action was performed at s. ϵ was chosen to be 0.15 within this exploration/exploitation scheme, which is the probability with which booQ chooses to exploit rather than explore. τ decays to ϵ, at which point booQ exploits with probability $1 - \epsilon$, hence the schedule name ϵ-decaying.

5.0 EXPERIMENTS

In this section, we evaluate the effectiveness of **(a)** our FIPS linear value model at judging our users based on their social metrics and **(b)** the booQ algorithm at selecting the correct user set for notifications. In the first experiment, we show the effectiveness of the baseline algorithms to motivate the importance of selecting users correctly, particularly with respect to minimizing food waste. In the second experiment, we investigate the sensitivity of the fairness metric to changes in the experimental configuration of the user history. In the third set of experiments, we demonstrate how adjusting various weights in the value model can improve performance on some metrics while compromising on others. The fourth experiment verifies the effectiveness of our problem formulation at avoiding frustration. In the fifth experiment, we compare the effectiveness of booQ at correcting user probability score bias to the baseline algorithms. Finally, we show how booQ learns a booking factor over time on biased datasets.

5.1 EXPERIMENTAL SETUP

Environment: Given that our application is still a work in progress at the time of writing, we could not perform experiments with real users. Additionally, a real environment would not allow us to perform true experimentation, given the large number of different variables, the inability to scale the number of users or number of servings, etc. Instead, we generated user and event datasets to run our experiments in a simulated environment. The event simulation environment iterates over the set of events, assigning users to each event with a priority determined by the chosen selection method (baseline, greedy knapsack, etc.).

Table 3: A summary of the parameters used in our experiments.

Parameters for Experiments	
Name	*Value*
Train iterations	1000
Test iterations	100
User dataset size	1000
Event dataset size	100
Bias range	$[-100\% \ldots 100\%]$

Parameters for Datasets	
Name	*Function*
Event servings	Truncated normal distribution $\mathcal{N}(40,\, 10^2)$ in range $[10 \ldots 70]$
User probability score	Truncated normal distribution $\mathcal{N}(0.5,\, 0.2^2)$ in range $[0, 1]$
User objective score	$\mathcal{N}(0,\, \sigma_m^2) + p$ in range $[0, 1]$ where σ_m is the standard deviation that produces the desired mean absolute error m and p is the user's probability score
User Pantry status	Random 10% set to true, 90% kept false
User history	$[0 \ldots 2^d - 1]_2$ where d is the length of the history

After estimating user attendance based on a hidden (to the system) true probability score, it evaluates its performance on the four metrics (average food waste, percent-went, total fairness, and average Pantry percentage).

Datasets: A user dataset consists of 1,000 users, each with four values: 1) a probability score, generated via a truncated normal distribution between 0.0 and 1.0 centered around 0.50 with a standard deviation of 0.20; 2) a Pantry value, which is 0 (false) for 90% of the users, and 1 (true) for a randomly selected 10% of users; 3) a recent history to represent fairness, which is a random binary string of 10 digits, indicating whether or not they were invited to the last 10 events (e.g., '1001111010', with the leftmost 1 meaning the user was notified of the most recent event); and, 4) an objective, or true, probability score, generated by applying slight gaussian noise to the user probability scores based on desired mean absolute error.

Training Vs. Test: We generate two datasets, one for testing and one for training, both with a mean absolute error between the users' probability score and objective score of around 0.125. We also create train and test event datasets of 100 events each, which are represented by a serving size generated using a normal distribution centered at 40 with a standard deviation of 10 and truncated to a value between 10 and 70.

Algorithms: Experiments using an algorithm that requires training, such as booQ, are run for 1,000 iterations over the event and user training datasets. All algorithms are then tested using 100 iterations over the test datasets to produce a set of final result metrics, calculated by the mean of the metrics over the test iterations. The baseline algorithms we use are:

1. **Random Sample**: select users at random until reaching capacity.
2. **Frequent-First**: select the users most probable to show up in-order until reaching capacity. This algorithm is expected to maximize *Percent Went*, i.e., minimize *frustration*.
3. **Round-Robin**: select users in a queued fashion, never notifying a user again prior to all others receiving a notification, until reaching capacity; performed over multiple events. This algorithm is expected to maximize *Total Fairness*.
4. **Pantry-First**: select the Pantry students first, followed by non-Pantry students arbitrarily, until reaching capacity. This algorithm is expected to maximize *Percent Pantry*.
5. **Greedy-Knapsack**: select the users in order of descending value until reaching capacity.

All baselines have two variants, one that counts users as a single serving, and one that uses the probabilistic weight constraint of the knapsack problem, as in equation (i) on page 15.

5.2 BASELINE COMPARISON (FIGURE 5)

Our first experiment compares the performance of our baseline algorithms on food waste, percentage of Pantry students, and total fairness. Figure 5 shows the results of this experiment. As you can see, the baseline algorithms perform as expected with respect to the metrics that they were proposed to optimize. Namely, both Pantry-First variants outperform the other algorithms for the Pantry metric, and both Round-Robin variants greatly outperform the others in fairness. Frequent-First performs poorly on both fairness and Pantry, and does not provide any benefits otherwise. Both variants of Random-Sample are relatively fairer than our other baselines but do not perform well on the Pantry metric.

A telling visual is the large discrepancy in average food waste between the two variants of a given algorithm. The variation which counts each user as a single serving wastes significantly more food than the version which utilizes the constraint from our knapsack formulation (i) on page 15 to count each user by their probability score. This speaks to the efficacy of framing this problem as the knapsack problem, in particular, using probability score as weight. The range of food waste when considering only the probability score variants of the baselines, on the other hand, is only about 2.5 servings, compared to a range of approximately 15 servings when considering the single serving variant. Although Greedy Knapsack performed comparably to Pantry First in general, the versatility allowed by parameter adjustments in the FIPS value model prompt further experimentation.

Figure 5 allows us to prune away a few of our selection algorithms. Clearly, given the large amount of food remaining after using the single serving variant of the algorithms, these selection methods were the first to be removed from consideration. Additionally, Frequent-First performed poorly on all of the metrics relative to the other algorithms, and Random-Sample was unable to select a reasonable portion of Pantry students; thus, we choose not to train booQ on these two algorithms in subsequent experiments. This leaves us with Greedy

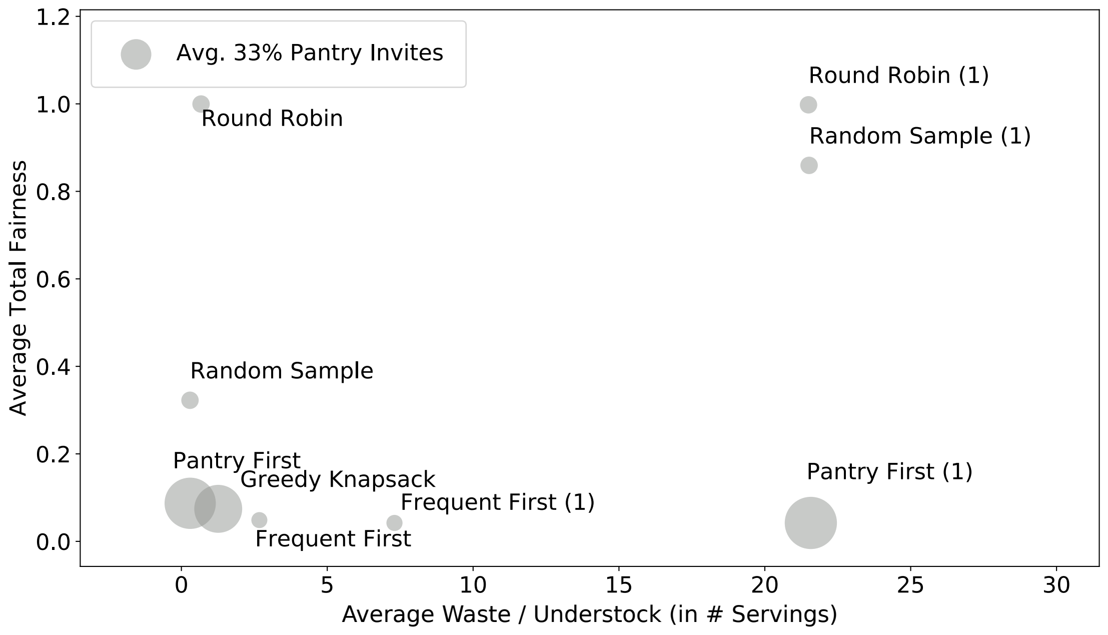


Figure 5: A comparison of the baseline algorithms. Positive waste is leftover food, whereas negative waste is an understock scenario. A suffix of (1) marks a baseline variant which counts each user as a single serving, as opposed to counting each user by their probability score. Each algorithm accumulates users until reaching capacity, defined as the number of servings S.

Knapsack, Round Robin, and Pantry First.

Lastly, Figure 5 illustrates the challenges of this problem: no one baseline selection method optimizes all of the desired metrics. In particular, fairness and Pantry are at odds with one another. This leads to our next experiment, exploration of the sensitivity of the FIPS value model with the Greedy-Knapsack selection.

5.3 FAIRNESS SENSITIVITY (FIGURES 6 & 7)

The second set of experiments examines the sensitivity of the fairness metric to changes in the experimental configuration of the user history. By adjusting both the initial values and length of the user invitation history, we learn the role of the history on calculating the fairness score.

First, we change how the user invitation history is initialized. Generally, the user history is randomized to "warm-start" the FIPS value. Here we compare our results using the greedy knapsack selection algorithm to an equivalent dataset with the user history initialized to zero, a.k.a., "cold-start". We use greedy knapsack because no other algorithm considers the FIPS value in selecting the user invitation set. As Figure 6 shows, there is virtually no difference in the fairness metric between the "warm-start" and "cold-start" initialization for user history. This is likely due to the fact that the history changes so quickly – the entire history is replaced after just 10 events.

Given the conclusion above, i.e., that the history is replaced too quickly for the choice in initialization to affect performance, we wanted to determine whether the length of the user invitation history would impact the performance on the fairness metric. For this experiment, we compare the baseline 10-event history with histories of length 25 and 100. A longer history means that a recent event invitation would yield an effect on the user fairness score for a longer period of time. In other words, it would increase the granularity of the fairness score.

The results of this experiment are shown in Figure 7 and Table 4. Although increasing the history length to 100 events slightly decreases the maximum average number of invitations received by any single user, it also slightly decreases the average number of invitations

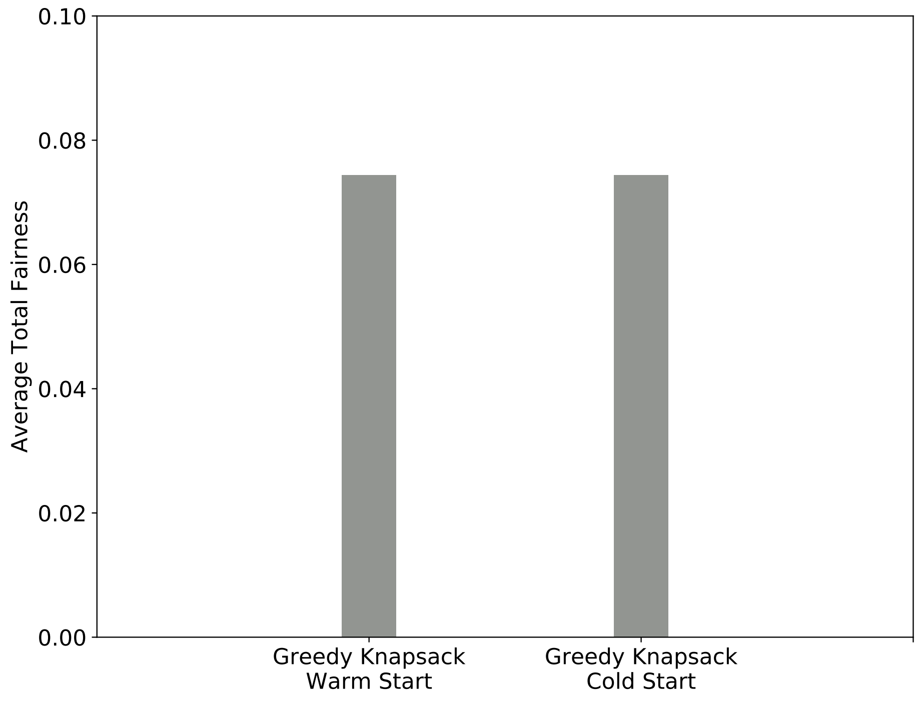


Figure 6: Comparison between warm-start (randomized) and cold-start (zero) invitation history initialization. The values indicate the starting value for the fairness score for each user and are replaced as event invitations are determined.

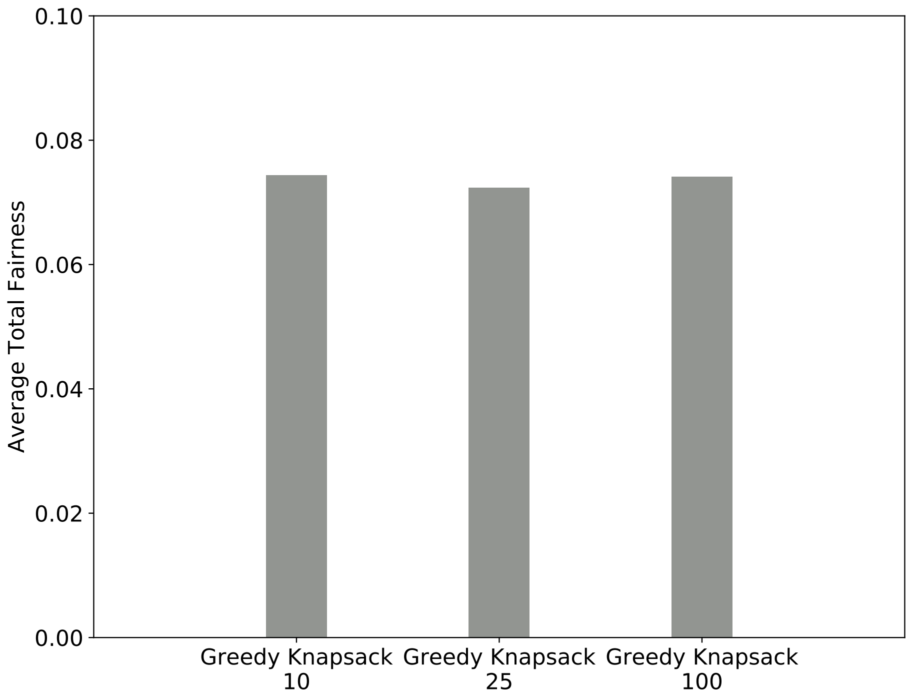


Figure 7: Comparison between invitation histories of various lengths. All histories begin with a randomized value and are replaced as event invitations are determined.

Table 4: Difference in event invitation distribution between various history lengths.

History length 10			
Max: 98.02	Min: 0	Mean: 7.29	Median: 0

History length 25			
Max: 97.84	Min: 0	Mean: 7.08	Median: 0

History length 100			
Max: 95.70	Min: 0	Mean: 7.09	Median: 0

received by the population. Table 4 clearly shows that the length of the history results in negligible change to performance on the fairness metric. This is likely due to the fact that a large number of users are still valued poorly by FIPS due to low probability scores and/or not being Pantry students in spite of the fact that a longer event history would cause recent event invitations to have a negative impact on the fairness scores of other users for a longer period of time. We can conclude that, in order to improve fairness by a considerable margin, we must specifically increase the weight of the fairness value in FIPS. Again, this improvement in fairness would come with a decline in food waste and Pantry performance.

5.4 FIPS VALUE MODEL SENSITIVITY (FIGURES 8 & 9)

This set of experiments shows the flexibility of the FIPS model by demonstrating its ability to adjust selection performance by altering the input weights. Unlike in Figure 5, the waste metric is not considered, since all of the considered weight combinations result in a negligible waste value between 0.0 and 2.5.

As you can see in Figure 8, incremental adjustments in value weight show a distinct

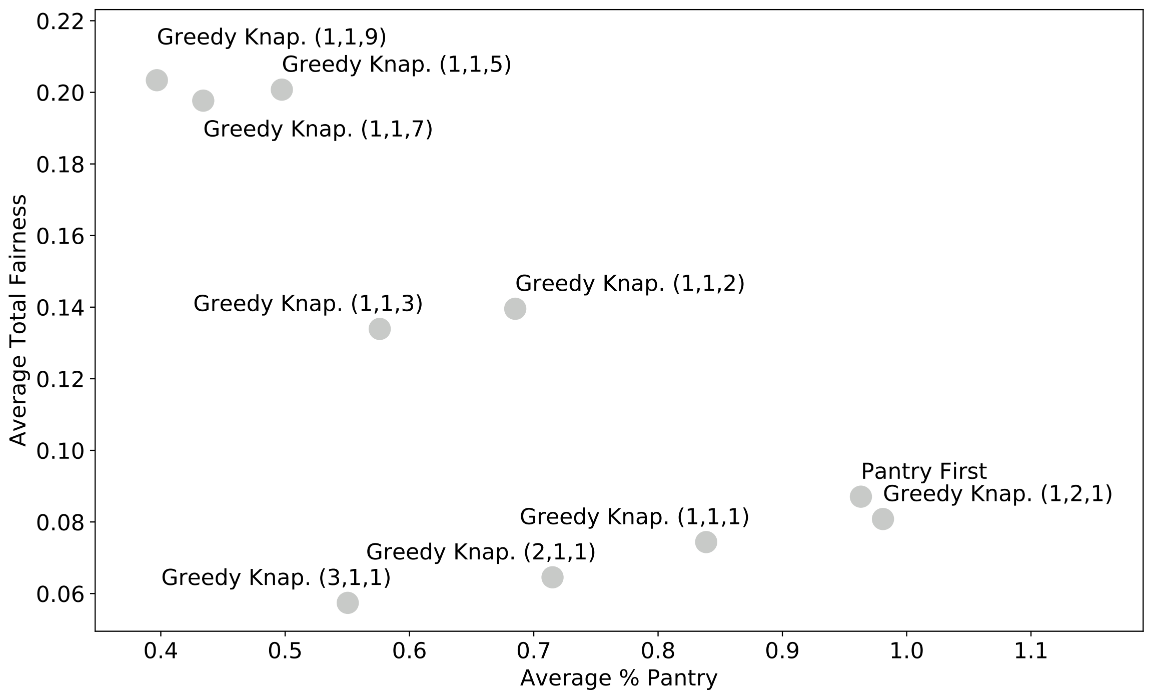


Figure 8: Sensitivity of FIPS model: the weights directly influence the ability of the selection algorithm to improve on the relevant metric. The order of the weights are as follows: probability score, Pantry, and fairness. This plot assumes an accurate probability score measurement from an unbiased estimator.

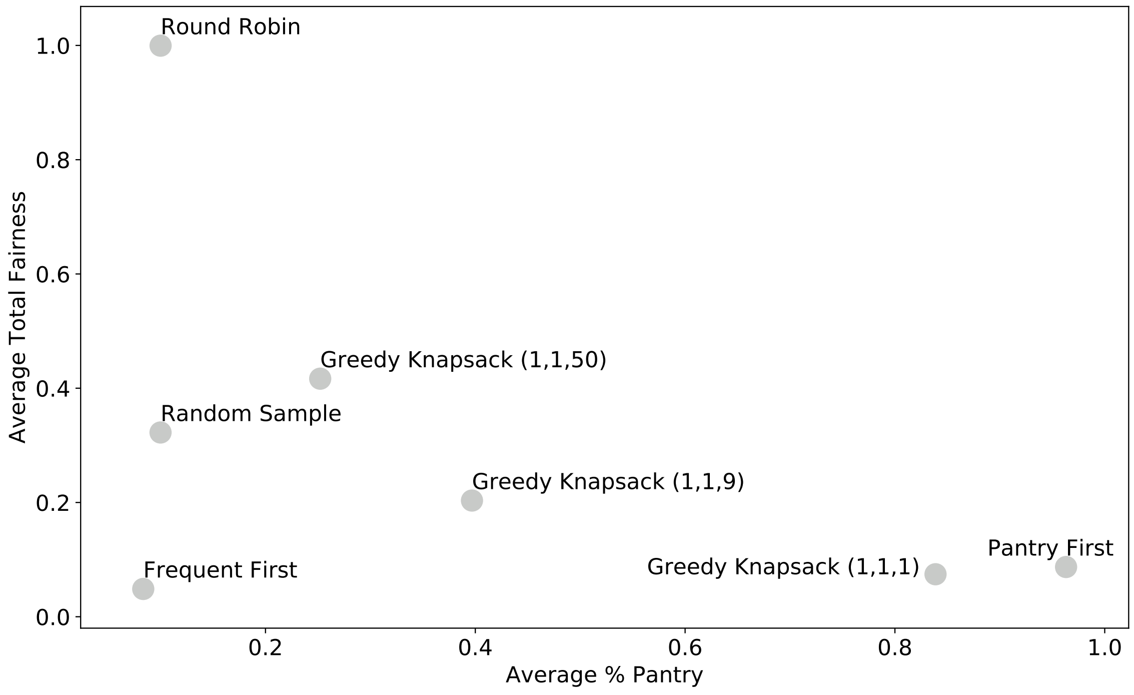


Figure 9: Sensitivity of FIPS model: various weights are compared to the baseline selection algorithms. This shows the effect that adjusting the parameters has on metric performance. The order of the weights are as follows: probability score, Pantry, and fairness. This plot assumes an accurate probability score measurement from an unbiased estimator.

trajectory of movement across the two dimensions of the plot: Pantry percentage and total fairness. For example, changing the fairness weight (the third weight) “moves” the Greedy-Knapsack point up, showing improved fairness metric performance. This illustrates the sensitivity of the model, again confirming the suitability of the knapsack problem with respect to user selection, in particular, the appropriateness of our FIPS linear value model.

Since our metrics have opposing goals, specifically fairness and Pantry competing with one another, increases in the fairness parameter result in a slight reduction in the algorithm’s performance on the Pantry metric. This is sensible, as the prioritization of fairness in a selection algorithm would de-prioritize special users, in this case, Pantry students. The trajectories seen in Figure 8 give us control over metric prioritization. This control should generalize to a higher-dimensional value model, i.e., FIPS+.

The sensitivity of the value model is further demonstrated in Figure 9, where a choice of parameters heavily preferring fairness results in a marked increase in average total fairness. While relatively high fairness is attainable with significant investment in the fairness parameter, it comes at a significant cost to performance on the Pantry metric.

5.5 FRUSTRATION PREVENTION (FIGURES 10 & 11)

One major consideration we made while designing this system was to avoid frustrating users. We estimate frustration using our percent-went metric, which calculates the percentage of users who went to an event out of all those that were invited. Sending invitations to users who do not attend the event is frustrating to them because they are receiving excessive notifications for events that, for one reason or another, do not interest them.

Figure 10 shows a comparison of the baseline algorithms, including greedy knapsack, with respect to percent-went in addition to our two social metrics. As expected, Frequent-First outperforms the other algorithms on percent-went by a large margin. On average, only about 20% of users did not attend the event for which they were invited. Also as expected, this baseline is severely outperformed on both fairness and percent Pantry by the others. Algorithm which avoids frustrating users must consider the users’ probability scores.

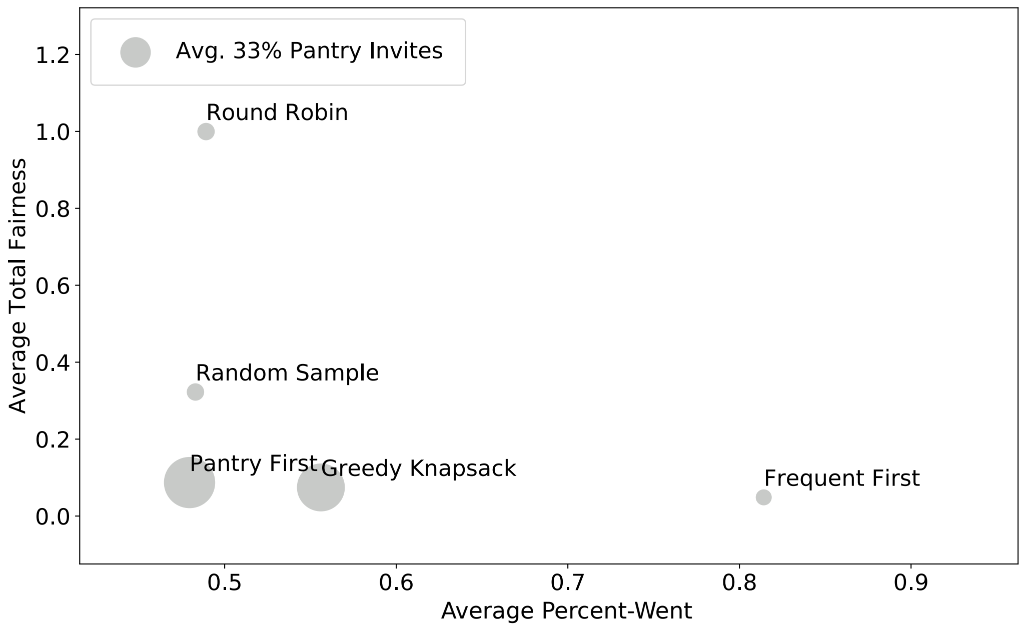


Figure 10: A comparison of the baseline algorithms focusing on performance on our metric for estimating frustration: percent-went. A large percentage of users that attend an event with leftovers relative to the number who were invited means that fewer uninterested or unavailable users were invited. Inviting users who do not attend an event is frustrating to the users due to receiving excess notifications.

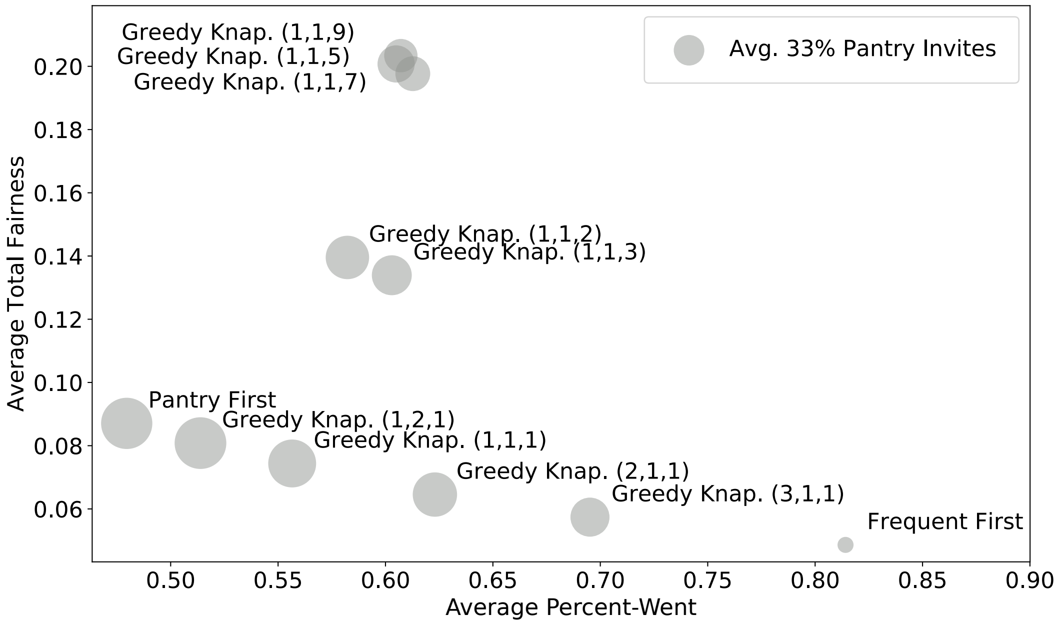


Figure 11: A comparison of the greedy knapsack selection algorithm with various FIPS parameter settings. The order of the weights are as follows: probability score, Pantry, and fairness. Increasing the probability weight improves performance on the percent-went metric, while increasing the other weights compromises it.

In Figure 11, we show the effect that an increase in the probability FIPS weight has on avoiding user frustration. With a minor investment in the probability weight, we are able to achieve a level of frustration similar to the Frequent-First baseline, while only minorly weakening performance in percent Pantry and fairness. Once more, we show the flexibility of the FIPS value model and the appropriateness of the 0-1 knapsack constraint formulation.

5.6 BIASED PROBABILITY EFFICACY (FIGURES 12, 13, 14, & 15)

A notable benefit of the knapsack formulation is the consideration of user probabilities as weights, as illustrated in experiments 5.2 and 5.5. One question, however, arises: what if the user probability scores are inaccurate, or highly biased? Moreso than fit, a biased probability estimator is a reasonable concern, as latent variables may cause a substantial directional error in probability estimates. This is a significant concern for our specific use case, because our target user demographic, being college students, is highly dynamic with respect to schedule and needs. For example, during the summer semester, when a majority of college students leave campus for home, our probability estimations could become critically inflated. Note that we are only interested in cases of directional bias in the probability estimates. Cases where the probability estimator is wrong but has no specific directional bias effectively exhibit a "cancel out" effect and do not lead to measurable changes in overall performance.

In this experiment, we illustrate how booQ resolves the problem of over- or under-estimation in user probability, which the baseline capacity criterion in equation (i) on page 15 alone does not accomodate. We adjust the user dataset by biasing the previously mentioned user test and train datasets by adjusting user probability scores by a multiplicative factor. As an example, multiplying the objective model by 0.5 results in an over-bias of 100% in our probability estimates.

As illustrated by Figures 12, 13, 14, & 15, which demonstrate over- and under-bias of various degrees, booQ is able to learn the correct booking factor to drive the food waste close to zero. When the bias was large, it performed similarly well as when the bias was

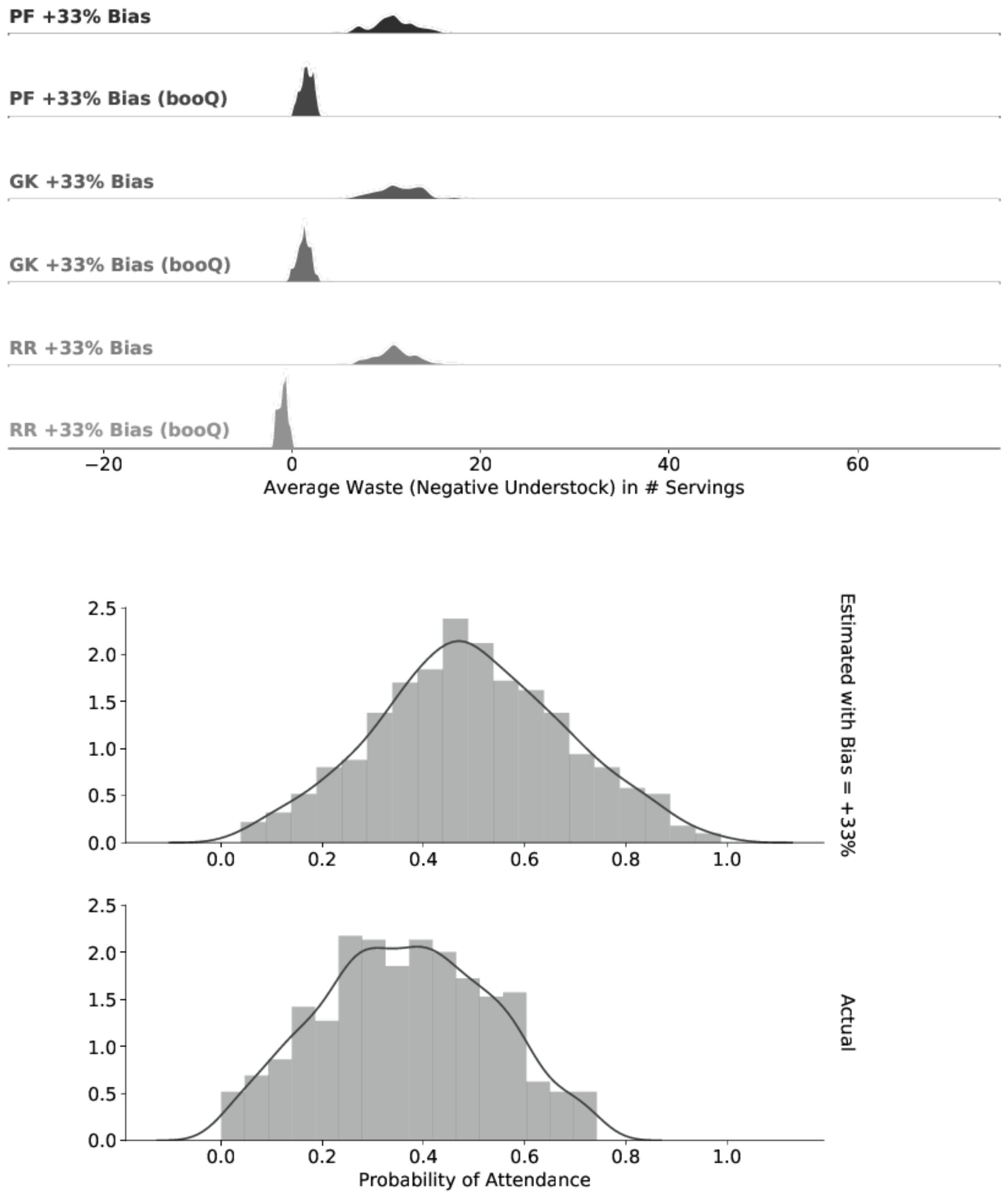


Figure 12: Selected baseline algorithms with and without booQ in the presence of probability estimation bias of 33%. booQ is able to learn and maintain minimal food waste in the presence of said bias. The lower plot shows the effect of the bias on the population's probability of attendance.

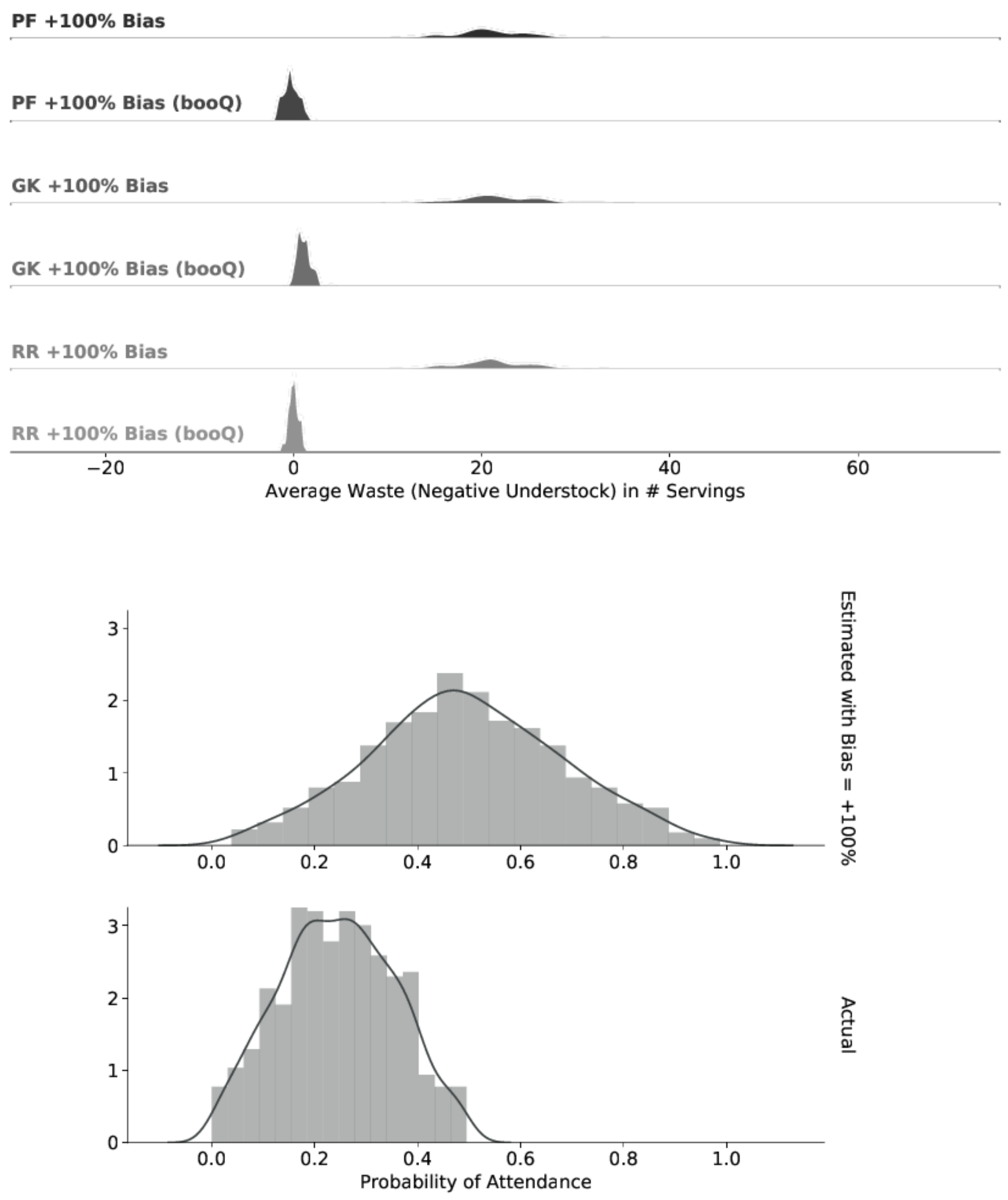


Figure 13: Selected baseline algorithms with and without booQ in the presence of probability estimation bias of 100%. booQ is able to learn and maintain minimal food waste in the presence of said bias. The lower plot shows the effect of the bias on the population's probability of attendance.

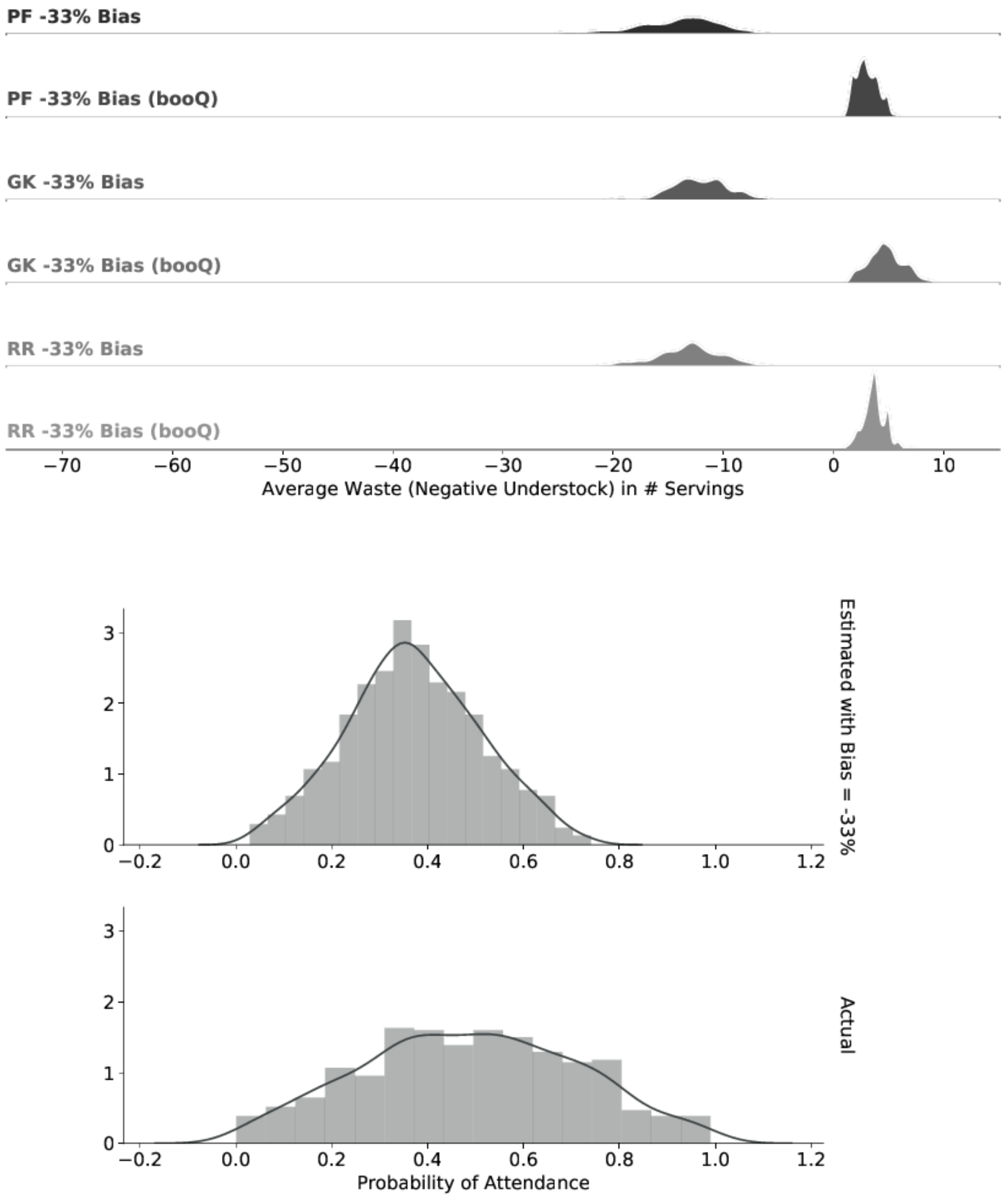
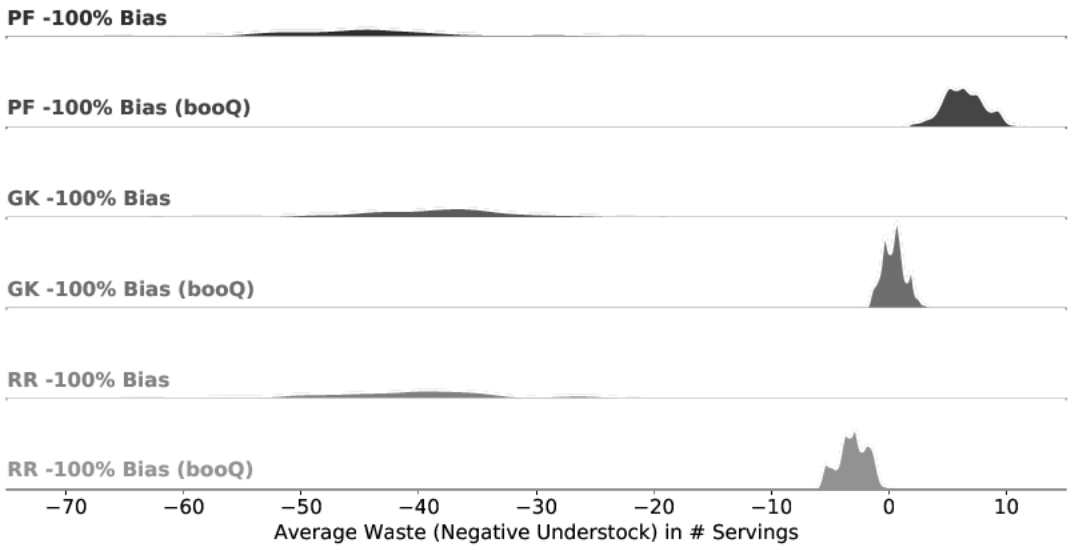


Figure 14: Selected baseline algorithms with and without booQ in the presence of probability estimation bias of −33%. booQ is able to learn and maintain minimal food waste in the presence of said bias. The lower plot shows the effect of the bias on the population's probability of attendance.



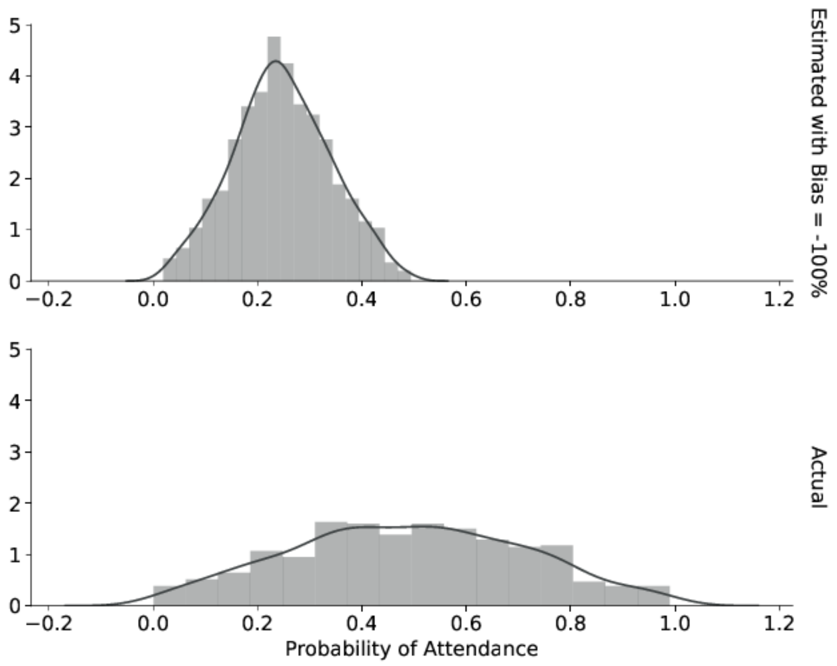


Figure 15: Selected baseline algorithms with and without booQ in the presence of probability estimation bias of -100%. booQ is able to learn and maintain minimal food waste in the presence of said bias. The lower plot shows the effect of the bias on the population's probability of attendance.

modest, indicating robustness. The other algorithms, including the baselines, were unable to effectively notify the students in these circumstances, resulting in significantly more food waste (or understock). The figures also show the significant improvement that booQ provides relative to the baseline algorithms. Biases between 33% and 100% are resolved by booQ in an equally effective manner. Note, while minimizing waste, booQ avoids any significant understock, maintaining a balance.

5.7 BIASED PROBABILITY LEARNING (FIGURE 16)

In Figure 16, we illustrate how booQ learns and adjusts ω over 1,000 training iterations using the Pantry-First selection algorithm. The dynamic step size results in booQ making larger adjustments to the booking factor when average waste is far away from zero, with a maximum single adjustment limited to 0.1. This allows booQ to alter suboptimal booking factors to an appropriate degree until converging upon a booking factor resulting in near-zero waste. When the booking factor is close to the optimal, the dynamic step size is limited to 0.05, precluding drastic jumps and facilitating fine-tuning. Even in scenarios where a series of bad decisions are made (caused by exploration and leading to a suboptimal booking factor), the dynamic step size allows booQ to promptly recover. On the population featuring less bias (i.e., plots on the third column of Figure 16), booQ is able to converge, and, in fact, does so in relatively few iterations.

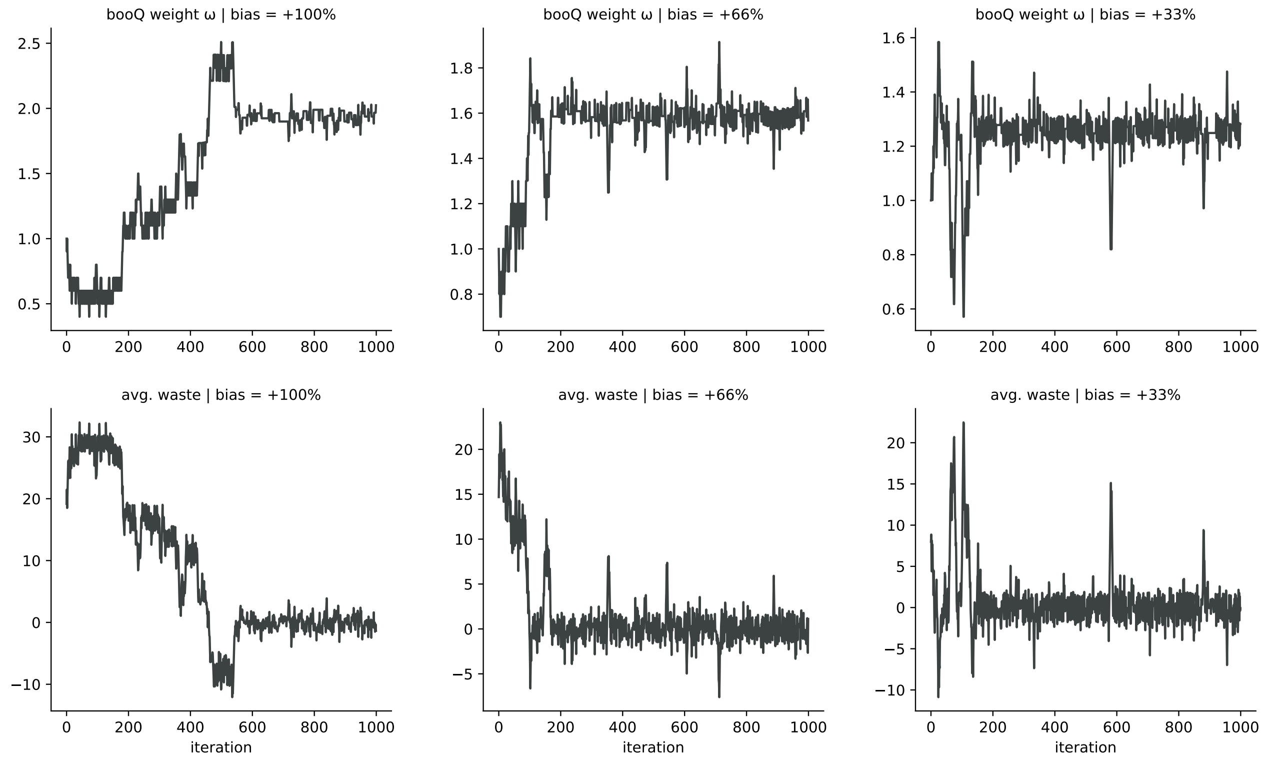


Figure 16: booQ learning on a positive probability bias with the Pantry-First selection algorithm. The dynamic step size allows booQ to promptly recover from poor decisions caused by exploration.

6.0 CONCLUSION

To reduce the amount of surplus food waste generated by institutions, we developed PittGrub. PittGrub provides a means of notifying University of Pittsburgh students of events with left-over food. Additionally, PittGrub is able to focus on two social metrics: food insecurity and fairness, which allows for prioritization of students facing difficult circumstances. This thesis describes PittGrub's smart notifications, consisting of:

1. **FIPS Value Model:** a model to value users based on their probability of attendance, fairness score, and food insecurity; FIPS includes tunable weights to prioritize specific attributes
2. **booQ:** a Q-Learning procedure to account for biased probabilty estimators and adjust user notification set size accordingly

We formulated the problem as a 0-1 knapsack problem and used the greedy knapsack algorithm to select a subset of users to notify based on their probability score (as a weight) and FIPS value. This formulation provides an appropriate way of prioritizing students who are facing food insecurity or have not recently been sent a notification from PittGrub.

Additionally, we developed a reinforcement learning algorithm based on Q-learning, booQ, which learns a multiplicative booking factor that, provided the number of leftover servings, determines the correct number of students to invite. The behavior of booQ can adapt to changes in circumstances such as bias in user probability of attendance.

A comprehensive experimental analysis shows the efficacy of FIPS parameter tuning for prioritizing specific user attributes and the adaptability of booQ with respect to learning the booking factor that estimates the notification set size in the presence of probability estimation bias. Compared to the defined baseline algorithms, FIPS and booQ are able to balance the performance on our social metrics while also ensuring a minimization of food that is left unserved. At the time of writing this thesis, the methodology described is currently being integrated into the PittGrub mobile application.

7.0 FUTURE WORK

In addition to our immediate next steps as outlined in Section 2.1, we intend to complete the following tasks in the near future:

1. Improve PittGrub's performance on the social metrics, especially fairness
2. Dynamically adjust notification volume based on rate of attendance

As shown in Section 5.4, the difficulty of improving PittGrub's performance on the fairness metric is that it comes at a cost to the performance on the food waste and Pantry metrics. We need to ensure that a proper weighing scheme is developed for the FIPS model such that the metrics can be properly balanced. Our first attempt at doing this was not very successful, but I want to share our work in order to lay a foundation for future improvements.

7.1 TRIVARIATE Q-LEARNING

Our first major attempt at creating a fully automated FIPS weighing scheme was to use what we learned from our work on the Q-learning algorithm booQ. Our aim was to have a system that, based on feedback from the user invitations, could learn the proper values for the FIPS weights that effectively balance the social metrics and minimize food waste. To this end, we developed another Q-learning algorithm, which we call trivariate Q-learning (TriQ for short), that learns by adjusting the weights one-at-a-time and learning from its performance on the metrics. This algorithm uses a similar approach as booQ (a univariate Q-learner), with the exception that the state space is three-dimensional. The TriQ learner adjusts a single weight per iteration (i.e., across the event and user training sets) and determines the effect that the weight adjustment had with respect to the metrics.

Unfortunately, with so many changing parts (history, weights, etc.), our TriQ learner did not perform consistently. Figure 17 shows the performance across the three metrics for five independent trials. The large variance in performance shows that the TriQ learner struggles with converging on a single set of weights. Figure 18 further demonstrates this inconsistency by showing the weight values as the TriQ learner modifies them for two independent trials. In this first plot, the TriQ algorithm maximizes fairness, while the second plot shows a clear preference for the Pantry weight.

Although our attempt to learn the FIPS weights was not successful, we aim to continue our work on the TriQ learning algorithm in hopes that we obtain more consistent results. In addition to Q-learning, we are contemplating other learning techniques that may be better suited to our circumstances.

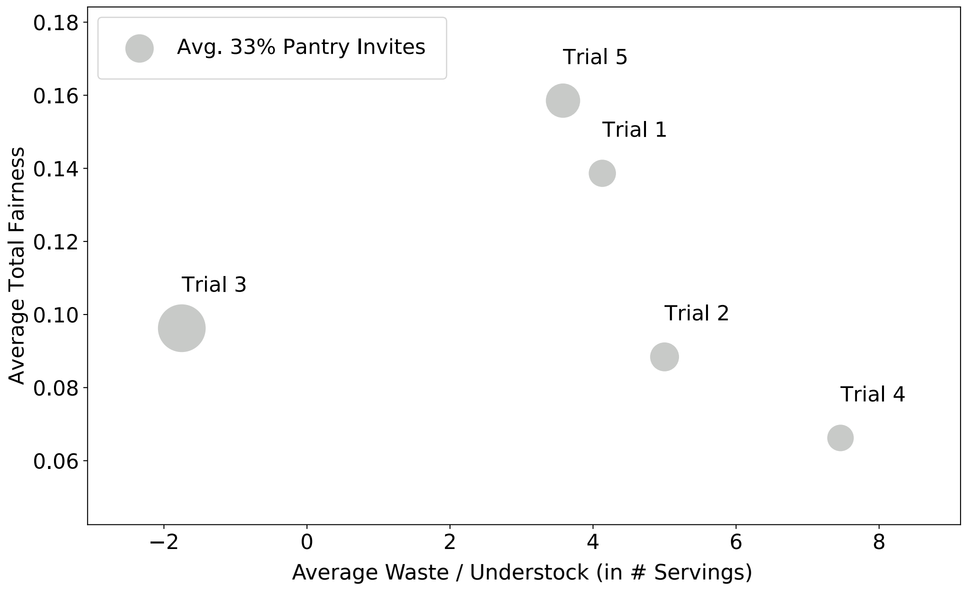
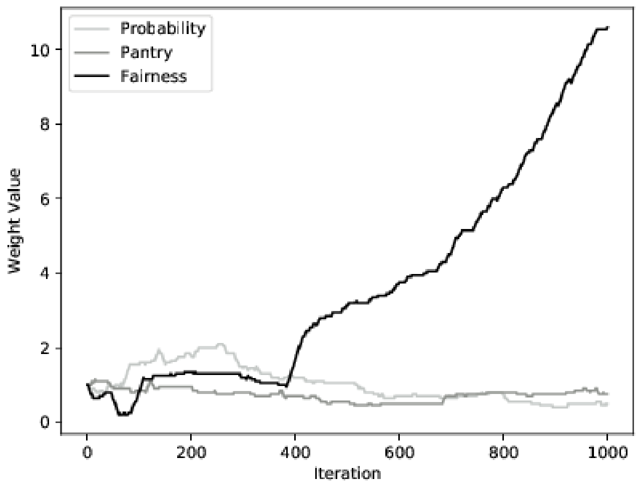


Figure 17: Performance comparison of five trials of the trivariate Q-learning implementation. The booking factor does not change while learning these weights. The inconsistency in the trial results indicates a difficulty in learning the correct weighing scheme.



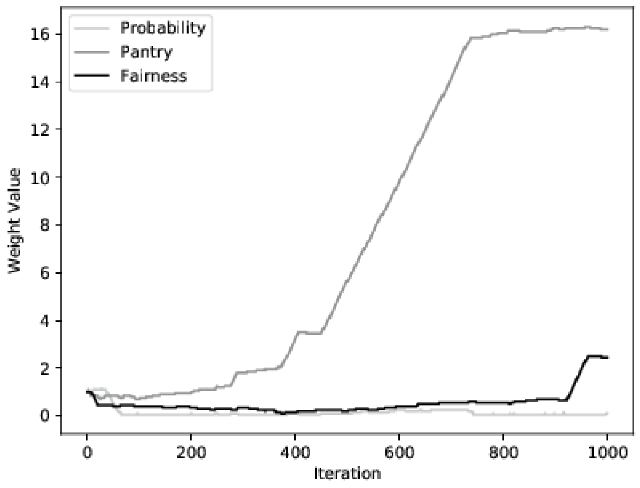


Figure 18: Example of the trivariate Q-learning implementation learning the weighting scheme. The two plots show a large discrepancy in weight selection. The upper plot decides to maximize the fairness weight, forsaking the others, while the lower plot maximizes the Pantry weight.

BIBLIOGRAPHY

[1] Environmental Protection Agency. https://epa.gov/sustainable-management-food/food-recovery-hierarchy, 2017.

[2] PittGrub App. https://expo.io/@admtlab/PittGrub, 2018.

[3] AWS Elastic Beanstalk. https://aws.amazon.com/elasticbeanstalk/, 2018.

[4] AWS Elastic Compute Cloud. https://aws.amazon.com/ec2/, 2018.

[5] Dana Gunders et al. Wasted: How america is losing up to 40 percent of its food from farm to fork to landfill. *Natural Resources Defense Council*, August 2017.

[6] Expo. https://expo.io, 2018.

[7] Joaquin Gonzalez. Pitt pantry helps students realize poor nutrition doesn't have to be part of the college experience. *WESA*, November 2017.

[8] Nick Goodfellow. Perch food waste audit report. https://www.pc.pitt.edu/dining/documents/Perchaudit2016FINAL.pdf, Spring 2016.

[9] Brandon Matthews James Dubick and Clare Cady. Hunger on campus: The challenge of food insecurity for college students. *National Student Campaign Against Hunger and Homelessness*, October 2016.

[10] Michael L. Littman Leslie Pack Kaelbling and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, May 1996.

[11] Silvano Martello and Paolo Toth. Knapsack problems. *Algorithms and Computer Implementation*, 1990.

[12] Matplotlib. https://matplotlib.org, 2018.

[13] React Native. https://facebook.github.io/react-native/, 2018.

[14] Food Recovery Network. www.foodrecoverynetwork.org, 2017.

[15] NumPy. http://www.numpy.org, 2018.

[16] PittGrub. https://pittgrub.com, 2018.

[17] PittServes. https://www.studentaffairs.pitt.edu/pittserves/sustain/pantry/, 2018.

[18] ReFED. A roadmap to reduce u.s. food waste by 20 percent. 2016.

[19] 412 Food Rescue. https://412foodrescue.org, 2018.

[20] Stuart Russell and Norvig Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd. edition, 2009.

[21] Scikit-learn. http://scikit-learn.org/stable, 2018.

[22] Seaborn. https://seaborn.pydata.org, 2018.

[23] AWS Relational Database Service. https://aws.amazon.com/rds/, 2018.

[24] AWS Simple Email Service. https://aws.amazon.com/ses/, 2018.

[25] CSUF News Service. Titan bites app helps students locate free food on campus. *CSUF News Center*, February 2017.

[26] SQLAlchemy. http://www.sqlalchemy.org, 2018.

[27] Tornado. http://www.tornadoweb.org/, 2018.