PittGrub: A Frustration-Free System to Reduce Food Waste by Notifying Hungry College Students

Mark Silvis
Department of Computer Science
School of Computing & Information
University of Pittsburgh
marksilvis@pitt.edu

Anthony Sicilia
Department of Computer Science
School of Computing & Information
University of Pittsburgh
anthonysicilia@pitt.edu

Alexandros Labrinidis
Department of Computer Science
School of Computing & Information
University of Pittsburgh
labrinid@cs.pitt.edu

ABSTRACT

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 placed in 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 paper, 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 a valuation model to determine *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 proposed techniques to multiple baselines on simulated datasets to demonstrate effectiveness. Experimental results among various algorithms show promise in eliminating food waste while helping those facing food insecurity and treating users fairly. Our prototype is currently in beta and coming soon to the Apple App Store.

CCS CONCEPTS

• **Information systems → Data mining**; • **Human-centered computing → Mobile devices**;

KEYWORDS

Reinforcement learning; Q-Learning; Data mining; Food waste

ACM Reference Format:
Mark Silvis, Anthony Sicilia, and Alexandros Labrinidis. 2018. PittGrub: A Frustration-Free System to Reduce Food Waste by Notifying Hungry College Students. In *KDD 2018: 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3219836

KDD 2018, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219836

1 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 [3]. Wasting such a large portion of our food is an enormous resource sink — we exhaust 19% of all farming fertilizer, 18% of all U.S. cropland, and 21% of U.S. agricultural water usage on the waste alone, totaling $218 billion (1.3% of the GDP) [14]. 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 [3].

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 [14]. Oftentimes, this waste consists of scraps that, although not fit for human consumption, can be used for animal feed or be composted rather than sent to a landfill; however, a large portion of wasted food is edible, yet simply discarded. An 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 [6]. 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 highest 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 [15]. Food Recovery Network is another non-profit which organizes student volunteers to collect and transport surplus food to food banks [10]. 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 a conference.

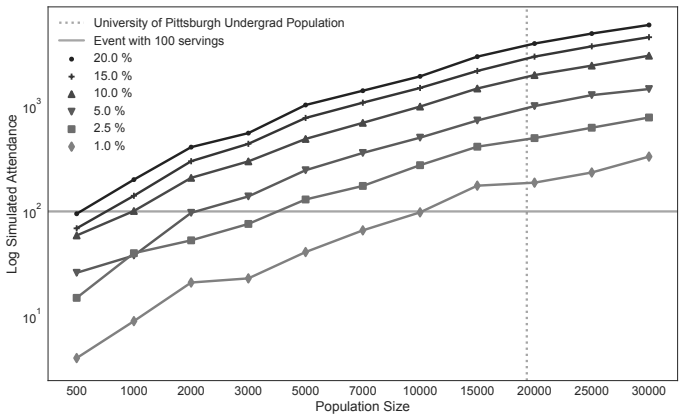

Figure 1: Logarithm of simulated attendance for user populations with varying size and varying average percentage of attendance. Here, the entire user base is notified of an event. For populations which are on par with PittGrub's potential audience of 19,326 undergraduates [11], simulations with very conservative choice of average percentage of attendance still lead to high user frustration due to over-booking.

Another approach to surplus food recovery is to have the people come to the food. An app called Titan Bites, 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 [17]. 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. Besides a lack of prioritization of food-insecure users, notifying the entire user base can lead to increased user frustration due to over-booking as the population scales. Figure 1 displays simulated attendance of users over varying population sizes with varying average percentage of attendance (1% - 20%). As the user base grows, mass notification can lead to poor user satisfaction, even under conservative assumptions of percentage of attendance.

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.

This paper's key contributions 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 the other way around. Additionally, we promote the humanitarian issues of food waste by prioritizing students in need, while also aiming for a fair notification spread 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, avoiding frustrating users by overbooking and wasting food by underbooking. 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 evaluating our selection of notification candidates that ensure 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.

2 PROTOTYPE

Although the sophisticated notifications part of our system is still under development, we have a working prototype of PittGrub available 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, approved by us, can post events with details like the number of 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. Figure 2 comprises screenshots of most of the current functionality.

PittGrub is available via Expo, a developer tool that runs apps built with the React Native framework on a local device without requiring Apple App Store distribution [5, 9]. As we are still in beta testing, distribution with Expo provides a fast and convenient way to send app updates to users. Our future plans for this project are to: 1) incorporate the methods from this paper into production, 2) finalize PittGrub's availability on the iOS App Store, and 3) aggressively promote PittGrub in order to gain traction; to that end, we have had multiple discussions with University Dining Services, the Student Office of Sustainability, and multiple student groups. At the time of the camera-ready, we have nearly 200 users signed up for the beta and a prototype on Test Flight at the iOS App Store. For

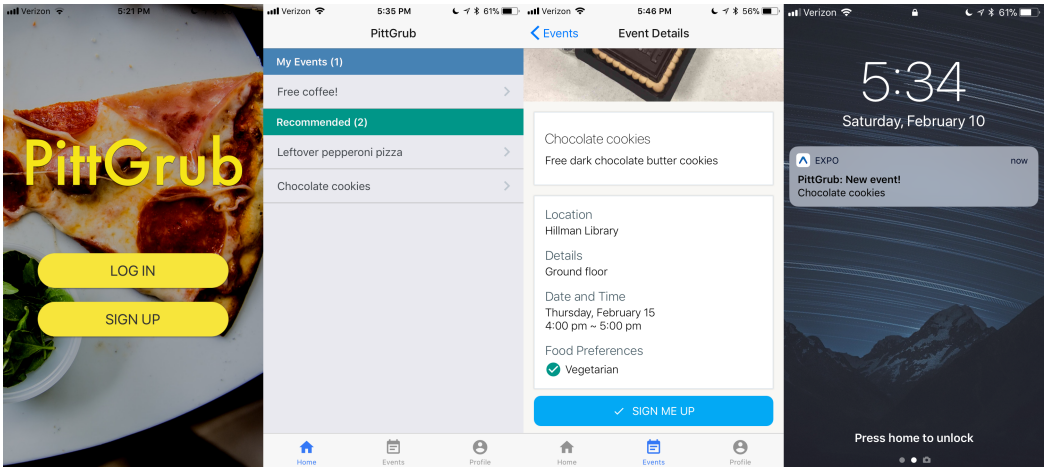

Figure 2: An overview of the current functionality of the PittGrub app. A user can log-in and view events, and approved users can post events. Users whose food preferences match those of a posted event will receive a notification (e.g., vegetarians will never receive notifications for non-vegetarian-friendly events).

the most up to date status, visit the PittGrub website [12] which includes news and installation instructions.

3 BACKGROUND

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

3.1 Pitt Pantry

The University of Pittsburgh has a program through the Student Office of Sustainability called Pitt Pantry, which helps to ensure that food-insecure students have regular access to healthy food options [13]. For the remainder of this paper, we will often refer to students facing food insecurity as members of the Pitt Pantry, or Pantry students for short.

3.2 Metrics & Baselines

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 minimization of food waste. In addition, we define three user-specific qualities that an ideal selection algorithm would be able to maximize:

(1) Average probability of user attendance per event
(2) Total notification fairness over a number of events
(3) Percentage of Pantry students notified per event

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) **Frustration:** defined as negative waste, i.e., we invite more users than there are servings; see Figure 1
(2) **Average Probability:** $\frac{\sum_{i=1}^{n_e} p_{i,e}}{n_e}$ where n_e is the number of notified users for event e and $p_{i,e}$ is the probability of attendance for user i and event e
(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 notified users for event e and $y_{i,e} \in \{0, 1\}$ is 1 for user i notified of event e if i is a Pantry student 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 simple averaging. For our later experimentation we focus on the use of aggregate metrics.

Using capacity to refer to the number of servings for an event, each of the previously described user metrics can be maximized using a specific baseline algorithm as follows:

(1) **Frequent-First**: select the users most probable to show up in-order until reaching capacity. This algorithm is expected to maximize *Average Probability*.
(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** section for comparison. There is no baseline for wasted food.

4 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 and never wasting 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* for each 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 paper, 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 valuates 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 valuation procedure can be calculated as: $\mathbf{v} = V(\mathbf{U}) = \mathbf{Uw}$, where $\mathbf{U}$ is a matrix with rows $u_i = (p_i, f_i, y_i)$ for each user; here p_i represents the aformentioned probability score, f_i represents a user's fairness score, and y_i their Pantry status with $\mathbf{w}$, a vector of weights 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:

$$f_i = 1 - \frac{h_{10}}{2^n - 1}$$

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 other's receive a 0. The history is then pruned to the 10 most recent events.

FIPS produces a linear valuation 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 valuate 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.

4.2 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 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 $O(nS10^d)$ 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 $O(n)$ time. On the same machine, this takes just over 1 minute to notify our 1000 user population of 100 various sized events. 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 [8]. 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 their FIPS value per event selection decisions. Since this requires re-sorting the users by their values for greedy selection, our greedy knapsack algorithm is actually $O(n \log n)$. 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).

4.3 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

Table 1: The rank of the different 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

possible that our blackbox probability estimator P has some directional error or bias which 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. In general, reinforcement learning consists of an agent which 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. 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 [7, 16].

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 negative frustration) 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 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 discernibly close to zero as possible. With regards to discernibility, the number of intervals determines the resolution of booQ after discretization; more intervals correspond to a higher resolution. For a more detailed analysis of this, see section 5.5. 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 each of these actions $a \in \{-1, 1, 0\}$. 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

Algorithm 1

procedure booQ:
  inputs:
    n: odd number of intervals τ: decreasing exploration rate
    α: decreasing learning rate ε: exploitation constant $\in [0, 1]$
    γ: 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 ω_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

corresponding to larger ρ

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

$$\rho = \frac{n+1}{2} \quad \text{otherwise.}$$

The reward, r, is then computed by observing differences between the current and 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 $r = \rho_t - \rho_{t-1} > 0$. But, if $\rho_t < \rho_{t-1}$ we have that 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 [7, 16]. The specifics of booQ can be found in Algorithm 1.

For choice of parameters, during experimentation, we took the discount factor to be $\gamma = 0.9$ [4]. A larger discount factor promotes prioritization of long-term rewards [7], a trait that is helpful in influencing booQ to take actions toward ideal ω that have been observed, but which are distant from the current ω. 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; see section 5.5 for sensitivity analysis. Table 1 represents the rank-space of booQ with $n = 21$. We chose a decreasing learning rate, α, as a function of the number of times the current state-action pair was seen [4]:

$$\alpha = \alpha\left(i_{s,a}\right) = \frac{1}{\left(i_{s,a}\right)^{\beta}} \quad \text{with} \quad \beta = 0.6$$

where $i_{s,a}$ is the number of times booQ took a at s. We chose τ in our $\varepsilon-$decaying exploration/exploitation scheme as a decreasing 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. In this decaying scheme, we took our lower bound $\varepsilon = 0.15$.

5 EXPERIMENTS

In this section, we evaluate the effectiveness of our FIPS linear value model at judging our users based on their social metrics and 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 set of experiments, we demonstrate how adjusting various weights in the value model can improve performance on some metrics while compromising on others. In the third 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, we cannot perform experiments with real users. Additionally, a real environment would not allow us to do 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, FIPS, FIPS with booQ, etc). 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, average user probability, 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 *Average Probability*.
(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 (i).

5.2 Baseline Comparison (Figure 3)

Our first experiment compares the performance of our baseline algorithms on food waste, percentage of Pantry students, and total fairness. Figure 3 shows the results of this experiment. As you can

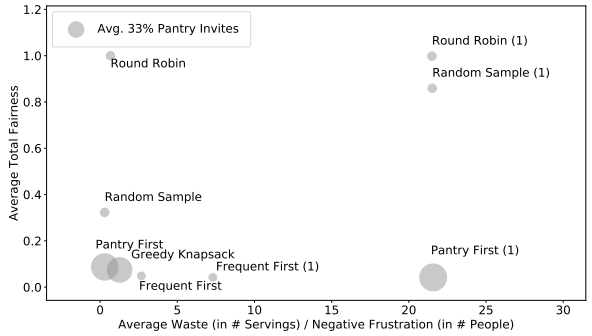

Figure 3: A comparison of the baseline algorithms. A suffix of (1) means the baseline counts each user as a single seat, as opposed to counting each user by their probability score. Each algorithm accumulates users until reaching capacity, defined as the number of servings S.

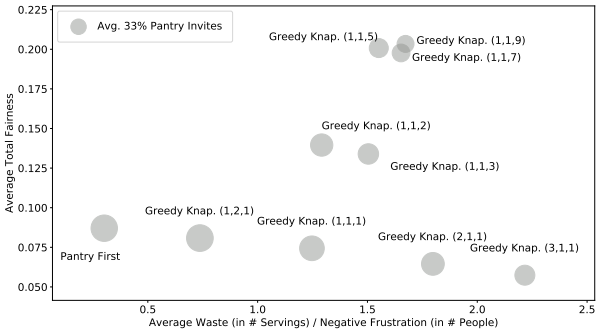

Figure 4: Sensitivity of FIPS model: the weights directly influence the ability of the selection algorithm to improve on the relevant metric. The order of the weights are as follows: probability score, Pantry, and fairness.

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 more fair than our other baselines but do not perform well on Pantry.

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 out knapsack formulation (i), i.e., probability score, as seat size. This speaks to the efficacy of framing this problem as the knapsack problem, in particular, using probability score as weight. 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 3 also allows us to prune away a few of our selection algorithms. Clearly, given the large amount of food remaining after using the single seating 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 following experiments. This leaves us with Greedy Knapsack, Round Robin, and Pantry First.

Lastly, Figure 3 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 FIPS Value Model Sensitivity (Figures 4 & 5)

This set of experiments shows the flexibility of the FIPS model by demonstrating its ability to adjust selection performance by

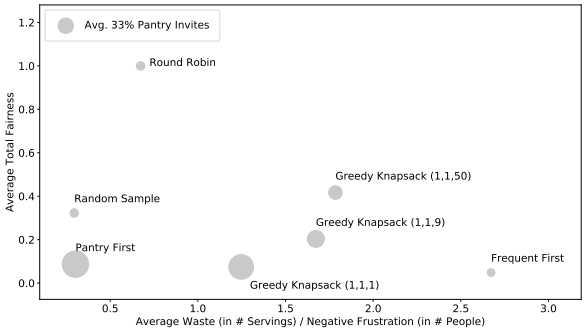

Figure 5: Sensitivity of FIPS model: various weights are compared to the baseline selection algorithms. This shows the effect of adjusting the parameters on metric performance. The order of the weights are as follows: probability score, Pantry, and fairness.

altering the input weights. As you can see in Figure 4, incremental adjustments in value weight show a distinct trajectory of movement across the three dimensions of the plot: wasted servings, total fairness, and Pantry percentage. For example, changing the fairness weight (the third weight) "moves" the Greedy-Knapsack point up. 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 valuation 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 4 give us control over metric prioritization. This control should generalize to a higher-dimensional value model, i.e., FIPS+.

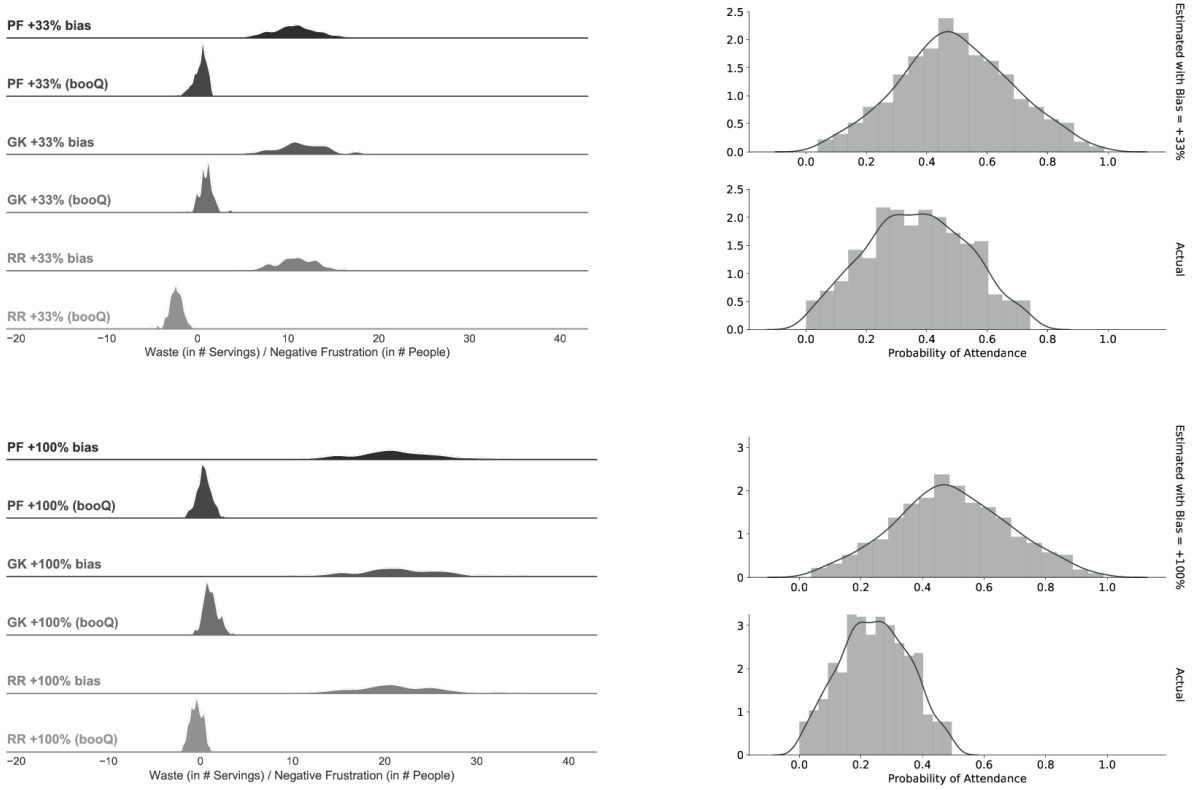

Figure 6: Left: Selected baseline algorithms with and without booQ in the presence of positive probability estimation bias. booQ is able to learn and maintain minimal food waste. Right: Visual representation of positive bias introduced to user probability estimation.

The sensitivity of the value model is further demonstrated in Figure 5, 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 cost to performance on the other metrics, especially waste and Pantry percentage.

5.4 Biased Probability Efficacy (Figures 6 & 7)

A notable benefit of the knapsack formulation is the consideration of user probabilities as weights, as illustrated in the first experiment. One question, however, arises: what if the user probability scores are inaccurate, or highly biased? More-so than fit, a biased probability estimator is a reasonable concern, as latent variables may cause significant 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. In this experiment, we illustrate how booQ resolves the problem of over- or under-estimation in user probability, which the baseline capacity criterion in (i) alone does not accommodate.

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 6 and 7, showing both over- and under-bias of various sizes, 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 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 user frustration). Figures 6 and 7 show the improvement that booQ provides over the baseline algorithms. Biases between 33% and 100% are resolved by booQ in an equally effective manner. Note that, while minimizing waste, booQ avoids any significant user frustration, maintaining a balance.

5.5 Biased Probability Learning and Parameters (Figure 8 & 9)

In Figure 8, we illustrate how booQ learns and adjusts ω over 1,000 training iterations using various selection algorithms. The dynamic step size results in booQ making larger adjustments to the booking

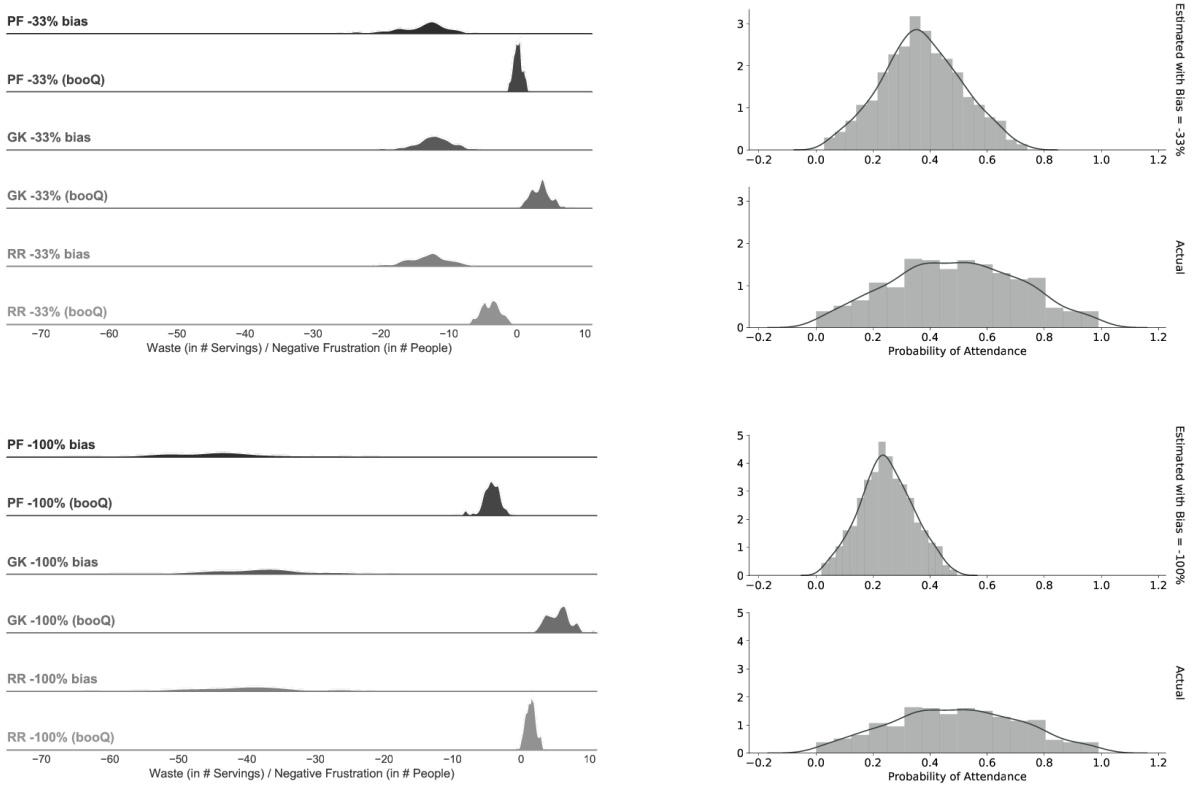

Figure 7: Left: Selected baseline algorithms with and without booQ in the presence of negative probability estimation bias. booQ is able to learn and maintain minimal food waste. Right: Visual representation of negative bias introduced to user probability estimation.

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 were made (caused by exploration and leading to a suboptimal booking factor), the dynamic step size allows booQ to promptly recover – Figure 8 exemplifies this in the case of Greedy Knapsack and Round Robin. In these scenarios, the larger discount factor $\gamma = 0.9$ may also contribute, allowing booQ to better account for distant rewards in its reward-update rule.

As mentioned in section 4.3, we chose the number of intervals $n = 21$ to increase booQ's resolution; Figure 9 demostrates this. Consistent convergence to an optimal booking factor occurs only for a larger number of intervals. This is likely due to increased feedback via reward and punishment during the learning process. Recall that the number of intervals corresponds to the number of states after discretization. A larger number of intervals partitions the state space more finely, allowing for smaller modifications of the booking constant ω to be recognizable and producing a greater breadth and diversity of rewards.

6 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 leftover food, focusing on food insecurity and fairness. This paper describes PittGrub's smart notifications, consisting of (1) **FIPS Value Model:** a model to valuate users on their probability of attendance, fairness score, and food insecurity; FIPS includes tunable weights to prioritize specific attributes and (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. A comprehensive experimental evaluation 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.

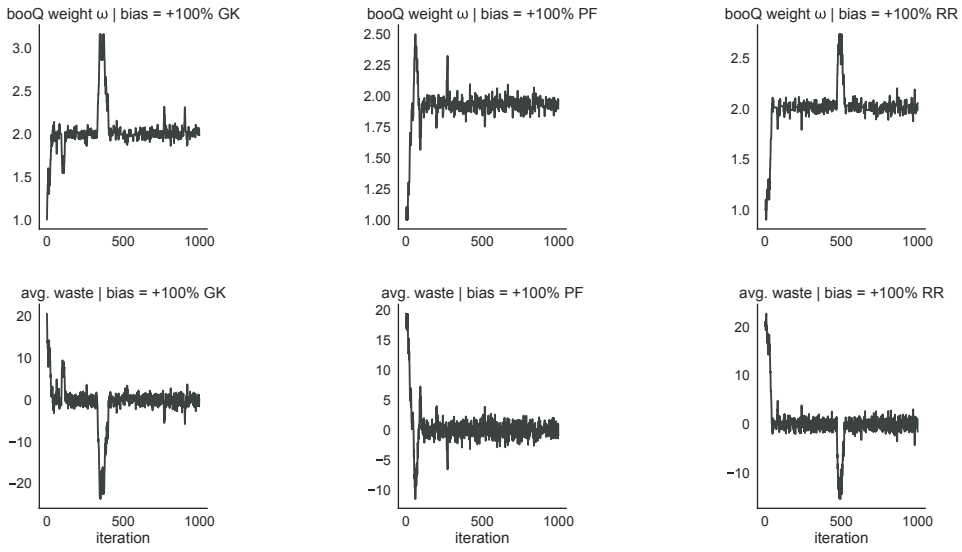

Figure 8: booQ learning on positive 100% probability bias. From left to right, the selection algorithms are Greedy Knapsack, Pantry First and Round Robin. The dynamic step size allows booQ to promptly recover from poor decisions caused by exploration.

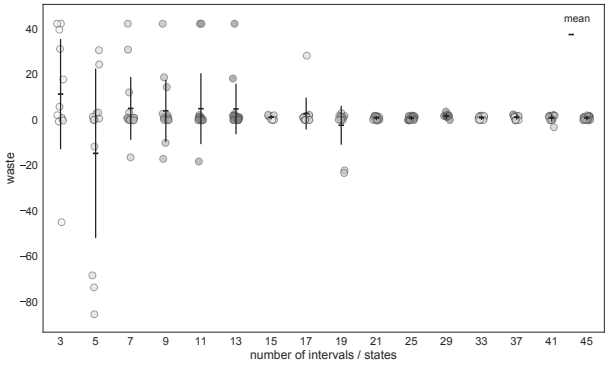

Figure 9: Analysis of booQ as the number of intervals, i.e, states, varies. Shown are waste measurements for 10 trials of each interval count with mean and standard deviation. Here, Greedy Knapsack is used as the selection algorithm. At a lower number of intervals, booQ becomes less consistent and often fails to find the optimal booking factor. booQ's consistency and resolution no longer suffer when the number of intervals is increased to 21 and greater.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their helpful suggestions on additional experimentation. This work was funded in part by NSF Award CNS-1739413.

REFERENCES

[1] Environmental Protection Agency. 2017. (2017). Retrieved February 09, 2018 from https://www.epa.gov/sustainable-management-food/food-recovery-hierarchy
[2] PittGrub App. 2018. (2018). Retrieved February 10, 2018 from https://expo.io/@admtlab/PittGrub
[3] Dana Gunders et al. 2017. Wasted: How America Is Losing Up To 40 Percent Of Its Food From Farm To Fork To Landfill. Report. *Natural Resources Defense Council* (Aug. 2017). Retrieved February 09, 2018 from https://www.nrdc.org/sites/default/files/wasted-2017-report.pdf
[4] Eyal Even-Dar and Yishay Mansour. 2003. Learning rates for Q-learning. *Journal of Machine Learning Research* 5, Dec (2003), 1–25.
[5] Expo. 2018. (2018). Retrieved February 09, 2018 from https://expo.io
[6] Nick Goodfellow. 2016. Perch food waste audit report. Report. (Spring 2016). Retrieved February 08, 2018 from https://www.pc.pitt.edu/dining/documents/Perchaudit2016FINAL.pdf
[7] Michael L. Littman Leslie Pack Kaelbling and Andrew W. Moore. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4, Article 301 (May 1996), 48 pages. https://doi.org/10.1613/jair.301
[8] Silvano Martello and Paolo Toth. 1990. Knapsack Problems. *Algorithms and Computer Implementation* (1990).
[9] React Native. 2018. (2018). Retrieved February 09, 2018 from https://facebook.github.io/react-native/
[10] Food Recovery Network. 2017. (2017). Retrieved February 08, 2018 from www.foodrecoverynetwork.org
[11] University of Pittsburgh Institutional Research. 2018. (2018). Retrieved May 24, 2018 from https://ir.pitt.edu/facts-publications/fast-facts/
[12] PittGrub. 2018. (2018). Retrieved February 10, 2018 from https://pittgrub.com
[13] PittServes. 2018. (2018). Retrieved February 08, 2018 from https://www.studentaffairs.pitt.edu/pittserves/sustain/pantry/
[14] ReFED. 2016. A Roadmap To Reduce U.S. Food Waste By 20 Percent. Report. (2016). Retrieved February 09, 2018 from https://www.refed.com/downloads/ReFED_Report_2016.pdf
[15] 412 Food Rescue. 2018. (2018). Retrieved February 08, 2018 from https://412foodrescue.org
[16] Stuart Russell and Norvig Norvig. 2009. *Artificial Intelligence: A Modern Approach* (3rd. ed.). Prentice Hall.
[17] CSUF News Service. 2017. Titan Bites App Helps Students Locate Free Food on Campus. *CSUF News Center* (16 Feb. 2017). Retrieved February 08, 2018 from http://news.fullerton.edu/2017wi/Titan-Bites-App.aspx