Rationing Experiences: Dynamic One-Sided Matching for Amusement Park Scheduling*

Lyle Goodyear† Ivan-Aleksandar Mavrov‡

January 23, 2025

Abstract

We describe practical, lightweight algorithmic solutions for the problem of one-sided matching with dynamic ordinal preferences and multiple capacities. In particular, we focus on amusement park scheduling where guests have preferences over rides they wish to experience and rides have limited capacity in each time period. We explore sequential variants of the Random Serial Dictatorship (RSD) and Probabilistic Serial (PS) mechanisms, and we experimentally validate their efficiency using a simulation calibrated to ride capacity and wait time data from Disneyland.

Keywords: Dynamic Matching, Ordinal Preferences, Market Design, Amusement Parks

*Final Paper for ECON 285: Matching and Market Design, Fall 2024

†Stanford University

‡Stanford University

1 Introduction

Nobody enjoys waiting in line. Amusement parks are entertainment destinations that feature various attractions, such as rides and games, each with a limited hourly capacity at any given time. Traditionally, amusement parks have rationed these limited experiences to their large number of guests using a simple physical queue system. Guests arrive at the attraction of their choice, they wait in line for a period of time determined by the service rate, then they experience the attraction.

The amusement park industry is a market experiencing consistent growth, with attendance numbers growing each year at the most-attended parks[1]. As parks become more crowded, we as market designers must more carefully consider how to ration the limited capacity of experiences on offer while simultaneously maximizing guest satisfaction and park revenue. Within the past few decades, major players in the amusement park industry have adopted alternatives to the standard physical queuing model to attempt to address the issue of growing congestion. We will focus on solutions offered by the Walt Disney Company at Disneyland in Anaheim, California.

As of December 2024, Disneyland currently offers 78 attractions including rides, shows, and other experiences[2]. Disneyland rations these experiences using four main methods. The first method is the traditional standby queue in which guests may join and wait for no additional cost. The second method is Lightning Lane Multi Pass, a one-time paid upgrade that offers the option to schedule a reservation time later in the day at select rides. Guests who purchase the Lightning Lane Multi Pass may book a reservation once per eligible attraction. Once they have booked a reservation, guests wait until their given time slot, at which point they arrive at the attraction and wait in a short queue with other Lightning Lane customers. Lightning Lane Multi Pass is typically offered for around 13 rides and excludes the most popular rides at the time.

[1] https://www.wdwmagic.com/attractions/epcot/news/15aug2024-disney-parks-dominate-2023-teaaecom-global-attendance-report,-epcot-sees-significant-growth.htm

[2] https://disneyland.disney.go.com/attractions/disneyland//sort=alpha/

The third method currently being used at Disneyland is Lightning Lane Single Pass, which offers a one-time reservation for one of the most popular rides. Typically, Lightning Lane Single Pass is offered for two attractions. Guests have to buy a separate reservation for each attraction in this category if they wish to use the Lightning Lane.

Finally, the fourth method is the virtual queue. Often, Disneyland will forgo standby queues entirely for its most popular attractions, opting instead for a virtual queue system. Twice per day, a limited number of virtual queue spots are released on the Disneyland app. Guests can claim a spot in the queue, subject to availability, after which they wait until their group is called to experience the attraction. Once called, they have one hour to walk to the attraction and redeem their place in the virtual queue. Attractions featuring a virtual queue will typically also have the option to purchase a Lightning Lane Single Pass, with which guests can select a specific reservation time.

The rationing system in place at Disneyland is stressful, unfair, and raises concerns about incentive compatibility. Virtual queue spots are given on a first-come-first-serve basis and are often completely gone within the first minute[3]. Guests with faster internet access, greater knowledge of the Disneyland app, or general tech-savviness have an unfair advantage in the process. Disneyland attracts a large number of foreign tourists who may have a particularly difficult time understanding the often-obscure instructions for navigating the current virtual queue system.

Another issue with the system is that it can dramatically inflate standby queue times. Ride operators are instructed to give priority to guests waiting in the Lightning Lane, meaning that the service rate for guests in the standby queue is much lower than the true hourly ride capacity. According to *Blog Mickey* in 2021, standby-to-priority service rate ratios can vary from 1:4 to a staggering difference of 1:10[4].

Finally, guests waiting in physical queues are a lost revenue opportunity. The advan-

[3]https://www.ocregister.com/2020/01/10/how-to-snag-a-spot-in-the-disneyland-virtual-queue-for-rise-of-the-resistance-at-star-wars-galaxys-edge/

[4]https://blogmickey.com/2021/11/disney-world-allocates-up-to-93-of-ride-capacity-to-lightning-lane/

tage of a virtual queue system is that guests are free to wander the park while waiting, incentivizing them to buy merchandise and spend money at restaurants.

We propose an alternative paradigm for rationing amusement park experiences that combines elements of reservation-based systems and virtual queues. Our system consolidates the rationing process into one central matching mechanism that allocates rides to guests at each time step. Informally, we model a day at the park as 16 discrete time steps (one hour each). At each time step, guests have ordinal preferences over the rides they wish to experience. We consider a dynamic setting, meaning that the preference list of each guest stochastically evolves according to the ride they were previously allocated (if any) and their previous preference list.

In this paper, we will describe practical, lightweight algorithmic solutions for the problem of one-sided matching with dynamic ordinal preferences and multiple capacities in amusement parks. We refer to this as the amusement park scheduling problem. In addition, we will provide an overview of the theoretical guarantees provided by each algorithm, summarizing prior literature. Finally, we will experimentally validate each algorithm using a simulated amusement park calibrated to capacity and wait time data from Disneyland.

2 Related Works

One-sided matching is one of the most fundamental and widely applicable fields of study in market design. One-sided matching markets can model a variety of practical resource allocation problems, such as assigning students to housing (Abdulkadiroğlu and Sönmez, 1998), matching individuals with job positions (Hylland and Zeckhauser, 1979), and allocating donor kidneys to patients in need (Roth, Sönmez, and Ünver, 2004). Famous algorithmic solutions to the one-sided matching problem include the Random Serial Dictatorship (RSD) mechanism (Abdulkadiroğlu and Sönmez, 1998) and the Probabilistic Serial (PS) mechanism (Bogomolnaia and Moulin, 2001). We will consider extensions of each to a dynamic setting.

It has long been known that *ex ante* Pareto efficiency and strategyproofness are incompatible with respect to cardinal preferences (Zhou, 1990). Bogomolnaia and Moulin, 2001 translate this result to settings with ordinal preferences. RSD represents a mechanism which is strategyproof but not ordinally efficient, while PS is ordinally efficient but sacrifices strategyproofness. However, in large markets where the number of copies of each item approaches infinity, Che and Kojima, 2008 showed that the allocations made by RSD and PS converge to each other. This result provides some hope for the efficiency of RSD in large markets.

Dynamic mechanism design is an extension of traditional mechanism design in a setting where the preferences of agents evolve over multiple time steps (Parkes, 2004). Surprisingly, there is little research on one-sided matching in dynamic settings with evolving preferences compared to theory on static matching. We therefore draw heavily from (Hosseini, Larson, and Cohen, 2015) for our model and theoretical properties, which describes two dynamic modifications of RSD. The first is Sequential RSD which sacrifices strategyproofness but runs in polynomial time during each matching period. The second is RSD with Adjusted Priorities (ARSD) which retains strategyproofness but runs in time $O(n!)$ (where n is number of agents). Due to its astronomical runtime, we will not cover ARSD in this paper. A practical solution to the amusement park scheduling problem should be lightweight, efficient, and easy to interact with from a user's perspective. While we will discuss strategyproofness, it is less important in this setting due to the limited time and resources available to agents who wish to manipulate the algorithm during each time period.

3 Model

In this section, we provide an overview of one-sided matching with dynamic ordinal preferences due to Hosseini, Larson, and Cohen, 2015. Additionally, we extend their model to include capacities that vary per object type.

Consider a market with $N = \{1, \ldots n\}$ agents (guests) and $D = \{1, \ldots, d\}$ objects (rides).

We assume that $n > d$. Additionally, each ride has a fixed capacity that resets during each matching period. We assume that the capacity is fixed throughout the day, therefore it does not depend on the period. We can think of the capacity as creating "copies" of each ride during that time period. We let the multiset $M = (D, m)$ denote the set of slots available on any ride, where $m : D \to \mathbb{N}_0$ is a multiplicity function that maps a ride $r \in D$ to its nonnegative integer capacity $m(r)$.

Guests have ordinal preferences over rides. We let $a \succ_i^t b$ mean that agent i strictly prefers ride a to ride b at time t. Note that guests' preferences are defined with respect to D, not the multiset M, that is guests only have preferences over the rides themselves and do not differentiate between the "copies" of each ride corresponding to its capacity. The set of all possible preference lists over D is denoted $\mathcal{P}(D)$, or $\mathcal{P}$, where $|\mathcal{P}| = d!$. We let $\succ^t = (\succ_1^t, \ldots, \succ_n^t)$ denote the *preference profile* of guests at time t, which contains the preference list of each guest.

A matching at time t, denoted $\mu^t : N \to D$ is a bijective mapping from guests to rides. The ride allocated to guest i at time t is denoted $\mu^t(i)$. A matching at time t is considered *feasible* if and only if for all rides $r \in D$, we have

$$\left|\left\{i : \mu^t(i) = r\right\}\right| \leq m(r), \forall r \in D \tag{1}$$

In other words, no ride should exceed its capacity. In the special case that all rides have capacity 1 ($m(r) = 1, \forall r \in D$), then the matching is feasible if and only if for all $i, j \in N$, $i \neq j$ implies $\mu^t(i) \neq \mu^t(j)$. During this section, we will assume this is the case, but we will give thought to multiple item capacities in sections 6 and 7. Furthermore, we impose the restriction that guests can be matched to at most one ride per time period. Similar to Hosseini, Larson, and Cohen, 2015, we model null assignments as "dummy rides" which no agent prefers to any other ride.

In our dynamic setting, the preference lists of each guest evolve during each time period.

Preferences evolve according to the history of previous realized matchings and preference profiles. In particular, we assume that the transition process is Markovian, thus the preference profile in time $t+1$ depends only on the matching μ^t and the preference profile $\succ^t$ in time t. We define a stochastic kernel $P(\succ^{t+1}|\succ^t, \mu^t)$ which determines the probability that agents will transition to a preference profile $\succ^{t+1}$ given realized matching μ^t and preference profile $\succ^t$. We denote the history up to time t as $h^t = (\succ^t, \mu^1, \ldots, \succ^{t-1}, \mu^{t-1}, \succ^t)$. Note that h^t includes the preference profile at time t.

Finally, we consider a matching policy $\pi(\mu \mid h^t)$ which gives the probability of applying the matching μ given the history h^t up to time t.

4 Sequential RSD

The Random Serial Dictatorship mechanism (RSD) is a canonical mechanism for one-sided matching in static settings (Abdulkadiroğlu and Sönmez, 1998). The mechanism operates by randomizing over *priority orderings* of agents, or permutations of a ranked list of agents. The mechanism then allocates the first ranked agent her first choice object, the second agent her highest ranked object with remaining capacity, and so on. RSD satsfies strategyproofness and *ex post* Pareto efficiency, but it is not ordinally efficient.

Sequential RSD is a mechanism which prescribes a sequence of RSD-induced matchings (Hosseini, Larson, and Cohen, 2015). In other words, at each time t, the mechanism picks a priority ordering of agents uniformly at random and allocates objects according to that ordering. To describe desirable properties of sequential RSD, we must first define *stochastic dominance*. Let

$$\mathcal{L}_i^t(r) = \{r \in D \mid x \succeq_i^t r\} \tag{2}$$

denote the lower contour set of ride r at time t under agent i's preferences. In other words, $\mathcal{L}_i^t(r)$ denotes the set of objects including r and all objects above r in agent i's preference

list. The probability of guest i being allocated ride r at time t is

$$p_i^t(r \mid h^t) = \sum_{\mu \in \mathcal{M}: \mu(i)=r} \pi(\mu \mid h^t) \tag{3}$$

where $\mathcal{M}$ denotes the set of all feasible matchings. Finally, the cumulative allocation probability for agent i at time t for a set $R \subseteq D$ is

$$\mathbb{P}_i^t(R) = \sum_{r \in R} p_i^t(r) \tag{4}$$

We say that a matching policy π stochastically dominates (sd) policy π' if

$$\mathbb{P}_i^t(\mathcal{L}_i^t(o^l)) \geq \mathbb{Q}_i^t(\mathcal{K}_i^t(o^l)), \qquad \forall i \in N, \forall l, \forall t \tag{5}$$

where $\mathbb{P}$ and $\mathbb{Q}$ are the cumulative allocation probability functions induced by π and π' respectively, $\mathcal{L}_i^t(o^l)$ is the contour set of the item ranked at position l in agent i's priority list at time t under policy π, and $\mathcal{K}_i^t(o^l)$ is the contour set of the item ranked at position l in agent i's priority list at time t under policy π'. In other words, policy π stochastically dominates π' if for each rank l, the probability that rides equal to or better than rank l are selected under π is greater than or equal to the same probability under π' (for every agent and time period).

A matching policy is *globally sd-strategyproof* (gsd-strategproof) if and only if truthfully reporting one's preferences is a stochastic dominant strategy for all possible realizations of history. Hosseini, Larson, and Cohen, 2015 provide an example demonstrating that sequential RSD is not globally sd-strategyproof. However, it is locally sd-strategyproof (lsd-strategyproof), meaning in each period, the mechanism is strategyproof with respect to the preferences in that period. This follows directly from the strategyproofness of static RSD. Additionally, the realized matchings satsfy *ex post* Pareto efficiency in each round, meaning we can represent the round as a uniform distribution of Pareto efficient outcomes. Finally, a

matching policy is *ordinally efficient* if it is not stochastically dominated by any other policy. Bogomolnaia and Moulin, 2001 prove that static RSD is not ordinally efficient, which again translates to per-round ordinal inefficiency for sequential RSD. In short, every property of static RSD applies locally to sequential RSD.

5 Sequential Probabilistic Serial

Bogomolnaia and Moulin, 2001 introduced the Probabilistic Serial (PS) algorithm for static random assignment, also known as the simultaneous eating algorithm. We first describe PS in its original implementation assuming the capacity of each item is 1. We note that Budish, Che, et al., 2013 prove that a generalized version of PS for items with multiple capacities retains the desirable efficiency and fairness properties of PS which we base our implementation on.

In the PS mechanism, we think of each ride as being one infinitely divisible item which is allocated among our n agents. If an agent i is allocated a proportion p_{ri} of ride r, this corresponds to the mechanism assigning ride r to agent with probability p_{ri}. In PS, agents "eat" their most preferred remaining ride simultaneously alongside other agents. We assume each agent has the same eating constant speed ω. The algorithm generalizes to variable eating speeds that may also differ among agents. The speed $\omega = 1$ represents the amount that each agent is allowed to eat per unit of time. We require that total amount each agent eats by the end of the algorithm (from $t = 0$ to $t = 1$) is one.

Given a preference profile, the algorithm lets each agent eat her favorite available item at speed $\omega = 1$ until that item is depleted. Once an item is depleted (the entire one unit has been allocated), all agents who were eating that item move to their next favorite available item. The algorithm proceeds in this manner until $t = 1$ or every item is depleted. The items are then allocated according to the probabilities determined by the proportion of each item eaten by each agent. For example, if agent i and j each consumed 0.5 of item r, then the

mechanism allocates r to agents i and j with probability 0.5 each. Once an item has been allocated, the item and agent are removed and the remaining probabilities are normalized.

We consider a sequential variant of PS which prescribes a sequence of assignments induced by PS. In each period, the mechanism runs PS with respect to the preference profile of that period. We claim that sequential PS retains the local ordinal efficiency of its static counterpart. Additionally, since static PS is not strategyproof, we claim that sequential PS is neither gsd-strategyproof nor lsd-strategyproof.

Finally, our practical implementation of PS needs to convert the generated probabilities into realized allocations that respect the capacities of each ride. We choose to iterate through all agents, greedily assigning available rides. When the capacity of a ride is fully allocated, we remove that ride and normalize the probabilities of the remaining rides for the remaining agents. We will discuss potential drawbacks of this implementation in sections 6 and 7.

Che and Kojima, 2008 demonstrated that the random assignments prescribed by static RSD and PS converge to each other as the number of copies of each object type approaches infinity. This implies that RSD is ordinally efficient "in the large". Their result provides rationale for the popularity of RSD for applications like student housing assignments. We experimentally interrogate the possible convergence of RSD to PS in each round in the following section.

6 Simulation

To better understand the performance of each of the aforementioned algorithms, we implement a simulation environment modeling a typical day at Disneyland. In 2022, the average number of daily visitors to Disneyland was 46,249[5]. We therefore create and update preference lists for 46,249 guests (assuming each guest is at the park for the entire day). We discretize time into 16 hour-long slots, representing the 16 hours that Disneyland is typically open for on a given day. Our simulated park includes 26 of the most popular rides at Dis-

[5]https://mickeyvisit.com/disneyland-statistics/

neyland in 2022 for which we could find capacity and wait time data, a representative but notably smaller subset of the 78 attractions currently on offer. We justify this discrepancy by noting that rides at Disneyland are often closed entirely for maintenance. Additionally, our simulation is a toy environment meant to demonstrate the capabilities of each algorithm, not a 1-to-1 digital twin of Disneyland. The ride capacities and average wait times are displayed in table 1.

Ride	**Average Wait**	**Capacity**
it's a small world	45	3300
Alice in Wonderland	45	600
Astro Orbitor	35	300
Autopia	20	3000
Buzz Lightyear Astro Blasters	40	1600
Casey Jr. Circus Train	5	360
Chip 'n' Dale's GADGETcoaster	30	780
Finding Nemo Submarine Voyage	35	850
Haunted Mansion Holiday	75	2620
Indiana Jones Adventure	65	2100
Jungle Cruise	30	1800
King Arthur Carrousel	15	1200
Mad Tea Party	20	500
Mark Twain Riverboat	10	300
Matterhorn Bobsleds	65	1700
Mickey & Minnie's Runaway Railway	40	2400
Millennium Falcon: Smugglers Run	45	1800
Mr. Toad's Wild Ride	25	654
Peter Pan's Flight	40	800
Pinocchio's Daring Journey	30	1200
Pirates of the Caribbean	45	3430
Roger Rabbit's Car Toon Spin	50	288
Space Mountain	45	2160
Star Tours – The Adventures Continue	20	1800
Star Wars: Rise of the Resistance	65	1700
The Many Adventures of Winnie the Pooh	35	2140

Table 1: Ride Information including average wait times and capacities. Wait times are averaged during 2019 and are taken from thrill-data.com. Ride capacities are taken from darkridedatabase.com.

In order to generate synthetic preference lists for our simulated guests, we sample from

a probability distribution over the rides where a higher probability corresponds to a higher popularity. In particular, we first normalize wait times and capacities using a discrete softmax function. That is, we create scores W_i and C_i for each ride (where w_i and $m(i)$ are the average wait time and capacity of ride i, respectively):

$$W_i = \frac{w_i}{\sum_j w_j}, \qquad C_i = \frac{m(i)}{\sum_j m(i)} \tag{6}$$

The overall popularity score for ride i is defined as $P_i = W_i \cdot C_i$. Intuitively, the popularity score balances the wait time and capacity (service rate) of each ride, ensuring that rides with high capacity and high wait times are marked the most "popular". We compute probabilities for each ride again using a discrete softmax function, that is, the initial probability of ranking ride i at position 1 for each agent is

$$p(o^1 = i) = \frac{P_i}{\sum_j P_j} \tag{7}$$

We sample without replacement for each agent, renormalizing the remaining probabilities as rides are selected to create a ranked preference list over all rides. The simulated preference profile $\succ^1$ at time $t = 1$ is summarized in figure 1.

Every simulation begins with the same initial preference profile, $\succ^1$. We consider four different underlying stochastic kernels. The first kernel P^1 implements static preferences, that is, $\succ^t = \succ^1$ for all times t. The second kernel P^2 implements uniformly random preferences, that is, $P^1(\succ^{t>1}) = \frac{1}{|D|!}$ irrespective of $\succ^t$ and μ^t. The third kernel P^3 implements random preferences in the same way as P^2, but the preference lists are instead generated using the computed popularity scores (the method we used for computing $\succ^1$). The fourth kernel P^4 implements a notion of how preferences might naturally evolve by ranking the most recent ride experienced by guest i at the bottom of i's preference list. Specifically, let $\succ^t_{i,-r}$ denote the preference list which has moved ride r from its ranking in $\succ^t_i$ to the bottom of the list, shifting each ride below r in $\succ^t_i$ up one position in $\succ^t_{i,-r}$. Then in kernel P^4, for each agent

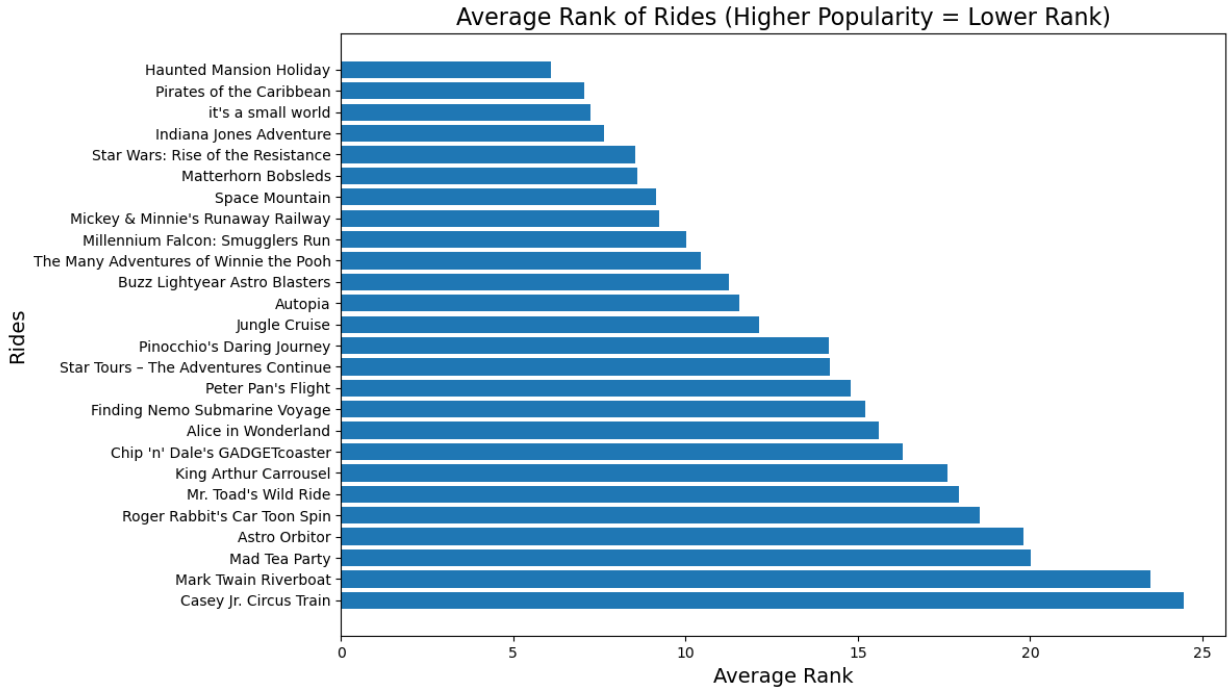

Figure 1: Average rank of each ride in the simulated preference profile $\succ^1$ at time $t = 1$.

i, their preference list will evolve according to

$$\succ_i^{t+1} = \begin{cases} \succ_{i,-r}^t & \text{if } i \text{ was allocated } r \text{ in time } t, \\ \succ_i^t & \text{if } i \text{ did not receive an allocation in time } t \end{cases} \tag{8}$$

6.1 Simulation Results

We tested sequential RSD and sequential PS on each of the four kernels. For each kernel, we created a plot displaying a histogram of the average rank of rides received across all rounds for each agent for both algorithms. In figure 2, which displays the histograms for kernel P^1, we can see that RSD allocates more mass in the highest rank. This makes intuitive sense, as each round is giving a different subset of agents their top ranked ride. The discrepancy between RSD and PS may also be explained by the implementation details of PS. We suspect our implementation of PS could be improved, particularly in how we convert the probabilities found by the mechanism to realized allocations. We normalized

the histograms for the purposes of comparison, but we observed that PS made slightly fewer allocations overall, likely due to the lottery system we configured.

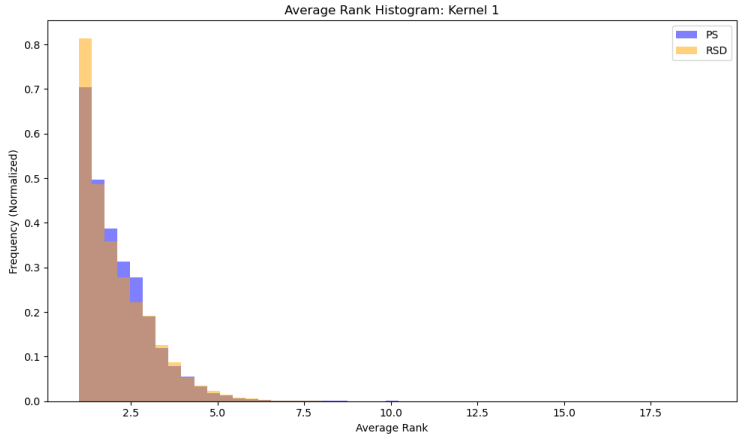

Figure 2: Superimposed histograms of average rank of allocation received by each agent across all rounds, kernel P^1

In figure 3, we can see that with uniformly random preferences, the average ranks are more normally distributed around the rank 3. The histograms are again very similar, with PS clustering more tightly around the mean. The histograms of allocations made under kernel P^3 tell a similar story, unsurprisingly.

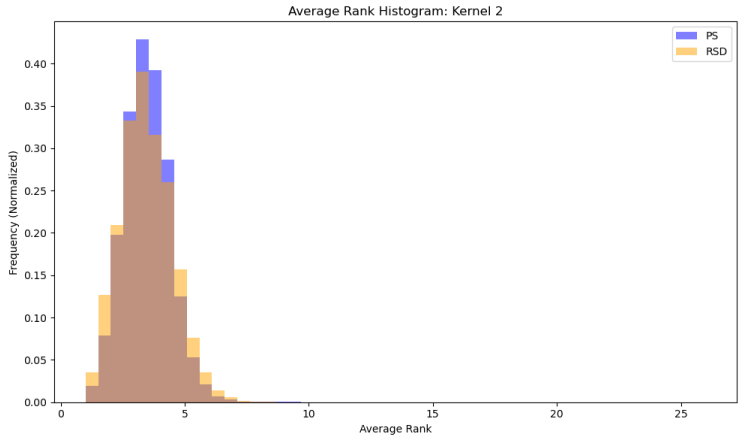

Figure 3: Superimposed histograms of average rank of allocation received by each agent across all rounds, kernel P^2

The results from kernel P^4 are more interesting, displayed in figure 5. We can see that RSD clearly has more mass on the highest rankings, suggesting that in this setting (which

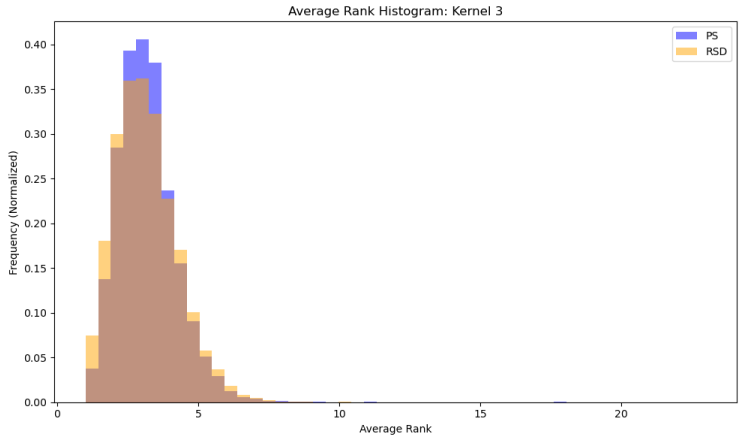

Figure 4: Superimposed histograms of average rank of allocation received by each agent across all rounds, kernel P^3

mirrors the real world more closely), RSD is the mechanism of choice.

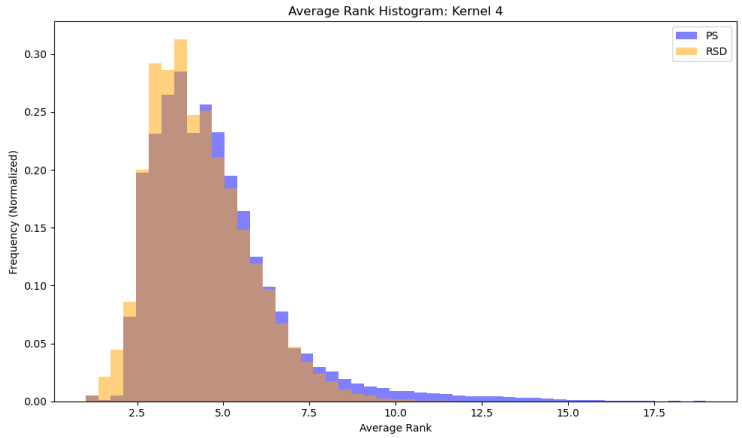

Figure 5: Superimposed histograms of average rank of allocation received by each agent across all rounds, kernel P^4

We also compared the expected allocations made by one-shot RSD and one-shot PS to better understand their possible convergence. We use the initial preference profile $\succ^1$ generated from the data. To infer the expected probabilities of agents being assigned each ride in RSD, we ran 1000 trials and computed probabilities from the empirical distribution of rides to each agent. For each agent, we computed whether the policy prescribed by PS weakly or strictly (stochastically) dominated, whether RSD weakly or strictly dominated, or whether neither mechanism dominated. We found that both mechanisms dominated for

near-equal proportions of the population, suggesting that the algorithms are converging in terms of efficiency. While more testing is required, we believe this result suggests that RSD is approximately ordinally efficient in markets of our size. This provides some rationale for the mechanism's popularity for applications like student housing assignment, a market with similar numbers of agents, goods, and capacities. The fraction of agents in each category is displayed in figure 6.

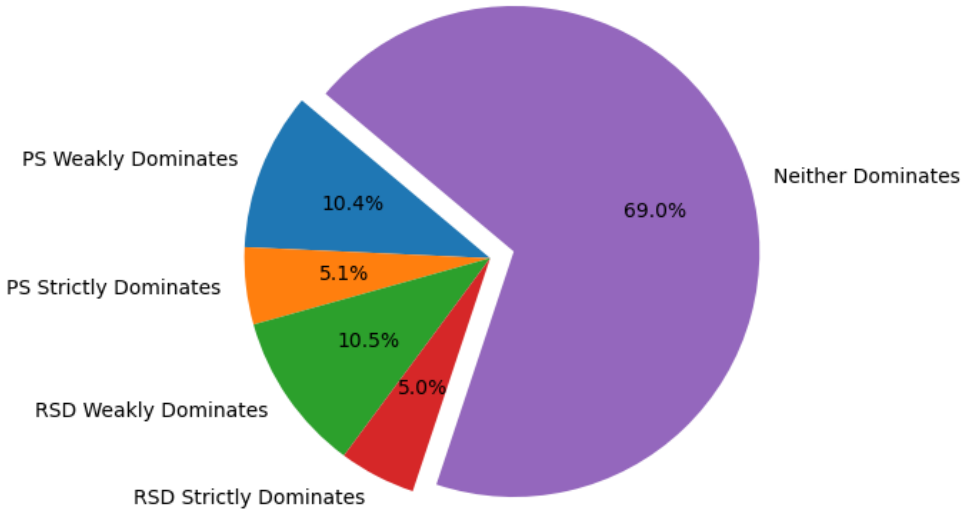

Figure 6: Fraction of agents for whom either RSD, PS, or neither dominated

7 Discussion

While more testing is required, our results seem to indicate that sequential RSD is a good fit for this setting. We showed that RSD and PS have similar efficiency, with RSD being much more computationally feasible. However, our implementation of PS likely introduces some inefficiency when randomly allocating rides. This could be due to the constraint we impose that ride capacities need to be respected. If we instead relax this constraint and provide lower capacities to the algorithm, we may recover some efficiency. Budish, Gao, et al., 2023 take a similar approach to handling capacities in their practical implementation of A-CEEI.

7.1 Limitations of the Model

Thus far, we have considered the performance of two different algorithmic solutions within our toy model. However, it is worth discussing where the model falls short of describing reality, as well as practical implementation details that factor in concerns about user experience and revenue.

Firstly, the model we consider is dynamic is the sense that preferences evolve stochastically, but it is not dynamic in the sense that guests enter and exit the market throughout the day. It would be worth examining a more general model which is able to depict guests entering and exiting the park throughout the day. Amusement parks often suffer from high congestion at peak hours, and such a model could provide insight on how to design a matching system to alleviate this congestion.

Furthermore, the amount of time a visitor spends at the amusement park may actually be correlated with the rides they are allocated. Certain visitors may have an internal "quota" of rides they wish to experience, after which they are ready to leave. Addressing such behavior would require further data analysis on patterns of behavior by current visitors. It could be possible to harness such behavior to better manipulate congestion and crowd flows.

A similar limitation of the model is that it cannot capture ride downtime, a frequent

issue at amusement parks which can greatly increase congestion at operating rides. It would be beneficial to be able to model downtime affecting certain rides throughout the day, in particular so we can update allocations when necessary to redistribute guests who were assigned to the non-functioning ride. In the same vein, it would be beneficial to introduce some variability in the capacity of each ride. Amusement park operators frequently vary the number of ride vehicles present in a ride to match the expected demand for that ride. Variable capacities would likely have a large impact on realized allocations in our model.

Finally, our toy model considered dividing the day into 16 time slots. A more realistic optimization framework would require a much greater level of granularity with respect to time. Such a framework would additionally need to take into consideration the constraints stemming from transit time between different locations within the amusement park as well as time for activities like eating at restaurants, shopping, or resting.

7.2 User Experience

What might it be like to interact with this system as a real guest at an amusement park? The system we propose is a departure from the status quo and could present challenges for guests to adapt to. We would like to design the system to mitigate these issues as much as possible. The most immediate hurdle is that guests may often want to deviate from their assigned ride. We can accommodate such deviations by reserving some amount of capacity for each ride to have a standby queue, but we will run into massive wait time inflation for those physical queues due to most of the capacity being allocated for virtual queue matching. Additionally, the more capacity we reserve for standby queues, the more efficiency we lose within the matchings made by RSD.

We could also consider treating the matching as a recommendation for the guest and utilize standby queues, allowing guests to freely choose rides they wish to experience while guiding them with recommendations. This is similar to the Genie system in use at Disneyland

currently which creates itineraries for guests based on their preferences[6] (and presumably tries to route them to minimize congestion park-wide). Genie does not require guests follow the itinerary it generates, nor does it reserve capacity at the recommended rides for those guests.

Submitting a full list of preferences over rides at each time period is cumbersome, and we should not expect guests to be willing or able to do this. Instead, we can require that they submit their preferences once at the beginning of the day, and we can update them at each time period according to the kernel P^4 (which moves their matching at time t to the bottom of their preferences in time $t+1$). Additionally, we can allow guests to edit their preference list whenever they like. Even with this relaxation, guests may still find it cumbersome to submit a ranked list of 78 rides once per day. We can instead allow them to rank as many as they want and infer the rest of the rankings from some notion of the popularity of the remaining rides.

7.3 Priority Upgrades

So far, we have not considered comparing revenue between our matching-based system and the status quo at Disneyland. Lightning Lane is a large source of revenue for Disney, so any system we design should preferably not eliminate that stream of revenue. As we have described it, our solution does not generate any additional revenue. However, we could consider offering one or more tiers of paid priority upgrades. These upgrades could function by splitting agents into different subsets by tier and running RSD first on the highest tier group, then on the next-highest tier group with the remaining rides, etc. This method could help recoup revenue that would be lost from eliminating Lightning Lane.

However, paid upgrades could also introduce concerns about fairness. In particular, depending on the number of upgrades sold, guests in the unpaid tier may not be allocated many rides at all. In some sense, this mirrors the situation at Disneyland currently, where

[6] https://disneyland.disney.go.com/genie/

guests who don't buy Lightning Lane might struggle to ride the most popular rides on a crowded day. However, guests might perceive our mechanism as being less fair; a central mechanism deciding that your child doesn't get to experience a popular ride "feels" less fair than the physical line simply being too long for you to wait in.

Additionally, it may be more difficult to convince visitors of the value offered by such paid upgrades. Currently, Lightning Lane is only offered if the rides will be available–a user makes a reservation, and as long as the ride doesn't experience downtime, they will be able to experience the ride. However, while our system guarantees high probabilities of being able to experience one's preferred rides, it cannot say for certain that a guest who pays for an upgrade will be get their money's worth. We could remedy this issue by providing statistical estimates of the benefits provided by the upgrade based on empirical allocations, but this is not as convincing as the promise made by Lightning Lane currently.

8 Conclusion

We have described the congestion problem facing amusement parks, the issues with current solutions, and a brief look at what a matching-inspired solution could look like. We provided an overview of literature on dynamic matching with ordinal preferences and implemented a toy simulation to test our algorithms. While our model is not perfect, we believe it offers insight into the practical use of sequential RSD for markets of similar size with dynamic preferences.

References

Abdulkadiroğlu, Atila and Tayfun Sönmez (1998). "Random Serial Dictatorship and the Core from Random Endowments in House Allocation Problems". In: *Econometrica* 66.3. Publisher: [Wiley, Econometric Society], pp. 689–701. ISSN: 0012-9682. DOI: 10.2307/2998580. URL: https://www.jstor.org/stable/2998580 (visited on 12/07/2024).

Bogomolnaia, Anna and Hervé Moulin (Oct. 2001). "A New Solution to the Random Assignment Problem". In: *Journal of Economic Theory* 100.2, pp. 295–328. ISSN: 0022-0531. DOI: 10.1006/jeth.2000.2710. URL: https://www.sciencedirect.com/science/article/pii/S0022053100927108 (visited on 09/25/2024).

Budish, Eric, Yeon-Koo Che, et al. (Apr. 2013). "Designing Random Allocation Mechanisms: Theory and Applications". en. In: *American Economic Review* 103.2, pp. 585–623. ISSN: 0002-8282. DOI: 10.1257/aer.103.2.585. URL: https://www.aeaweb.org/articles?id=10.1257/aer.103.2.585 (visited on 12/08/2024).

Budish, Eric, Ruiquan Gao, et al. (May 2023). *Practical algorithms and experimentally validated incentives for equilibrium-based fair division (A-CEEI)*. en. arXiv:2305.11406 [cs]. URL: http://arxiv.org/abs/2305.11406 (visited on 10/29/2024).

Che, Yeon-Koo and Fuhito Kojima (Oct. 2008). *Asymptotic Equivalence of Probabilistic Serial and Random Priority Mechanisms*. en. SSRN Scholarly Paper. Rochester, NY. URL: https://papers.ssrn.com/abstract=1277220 (visited on 11/23/2024).

Hosseini, Hadi, Kate Larson, and Robin Cohen (Feb. 2015). "Matching with Dynamic Ordinal Preferences". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 29.1. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v29i1.9329. URL: https://ojs.aaai.org/index.php/AAAI/article/view/9329 (visited on 11/22/2024).

Hylland, Aanund and Richard Zeckhauser (Apr. 1979). "The Efficient Allocation of Individuals to Positions". In: *Journal of Political Economy* 87.2. Publisher: The University of Chicago Press, pp. 293–314. ISSN: 0022-3808. DOI: 10.1086/260757. URL: https://www.journals.uchicago.edu/doi/abs/10.1086/260757 (visited on 09/25/2024).

Parkes, David C. (2004). "On Learnable Mechanism Design". en. In: *Collectives and the Design of Complex Systems*. Ed. by Kagan Tumer and David Wolpert. New York, NY: Springer, pp. 107–131. ISBN: 978-1-4419-8909-3. DOI: 10.1007/978-1-4419-8909-3_3. URL: https://doi.org/10.1007/978-1-4419-8909-3_3 (visited on 12/08/2024).

Roth, Alvin E., Tayfun Sönmez, and M. Utku Ünver (May 2004). "Kidney Exchange*". In: *The Quarterly Journal of Economics* 119.2, pp. 457–488. ISSN: 0033-5533. DOI: 10.1162/0033553041382157. URL: https://doi.org/10.1162/0033553041382157 (visited on 12/07/2024).

Zhou, Lin (Oct. 1990). "On a conjecture by gale about one-sided matching problems". In: *Journal of Economic Theory* 52.1, pp. 123–135. ISSN: 0022-0531. DOI: 10.1016/0022-0531(90)90070-Z. URL: https://www.sciencedirect.com/science/article/pii/002205319090070Z (visited on 12/08/2024).