Learning Riders' Preferences in Ridesharing Platforms

Ritaban Bhattacharya
rb5344@rit.edu

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Learning Riders’ Preferences in Ridesharing Platforms

by

Ritaban Bhattacharya

A Thesis presented in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Department of Computer Science

Golisano College of Computing and Information Sciences

Rochester Institute of Technology

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Hadi Hosseini, Advisor
Department of Computer Science
Rochester Institute of Technology

Dr. Zachary Butler, Reader
Department of Computer Science
Rochester Institute of Technology

Dr. Stanislaw Radziszowski, Observer
Department of Computer Science
Rochester Institute of Technology
Learning Riders’ Preferences in Ridesharing Platforms

by

Ritaban Bhattacharya

THESIS
Presented to the Department of Computer Science
of the Golisano College of Computing and Information Sciences
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science in Computer Science

Rochester Institute of Technology
May 2020
Abstract

Learning Riders’ Preferences in Ridesharing Platforms

Ritaban Bhattacharya, M.S.
Rochester Institute of Technology, 2020

Ridesharing platforms allow people to commute more efficiently. Ridesharing can be beneficial since it can reduce the travel expenses for individuals as well as decrease the overall traffic gridlocks. One of the key aspects of ridesharing platforms is for riders to find suitable partners to share the ride. Thus, the riders need to be matched to other riders/drivers. From the social perspective, a rider may prefer to share the ride with certain individuals as opposed to other riders. This leads to the rider having preferences over the other riders. A matching based on social welfare indicates the quality of the rides. Our goal is to maximize the social welfare or the quality of rides for all riders. In order to match the riders, we need to know the preferences of the riders. However, the preferences are often unknown.

To tackle these situations, we introduce a ridesharing model that implements reinforcement learning algorithms to learn the utilities of the riders based on the riders’ previous experiences. We investigate a variety of measures for assessing social welfare, including utilitarian, egalitarian, Nash, and leximin social welfare. Additionally, we also compute the number of strong and
weak blocking pairs in each socially optimal matching to compare the stability of these matchings. We provide a comparison between two reinforcement learning algorithms: $\epsilon$-greedy and UCB1, for learning utilities of the riders, maximizing social welfare, and the number of blocking pairs in the socially optimal matching.

The $\epsilon$-greedy algorithm with $\epsilon = 0.1$ provides the maximum accuracy in learning the utilities of the riders as compared to $\epsilon = 0.0$, $\epsilon = 0.01$, and UCB1 algorithm. It also provides fewer number of blocking pairs suggesting more stability in the socially optimal matching than other reinforcement learning algorithms. However, UCB1 algorithm outperforms all other reinforcement learning algorithms to provide maximum welfare in the socially optimal matchings.
# Table of Contents

Abstract iii

**Chapter 1. Introduction** 1

1.1 Introduction .................................................. 1
1.2 Contributions ................................................. 3

**Chapter 2. Related Works** 6

**Chapter 3. Model** 10

3.1 Model .......................................................... 10
3.2 Social Welfare Functions .................................... 12
    3.2.1 Utilitarian Social Welfare .......................... 13
    3.2.2 Egalitarian Social Welfare ......................... 14
    3.2.3 Nash Social Welfare ................................. 15
    3.2.4 Leximin Social Welfare ............................. 16
3.3 Stable Matching ............................................. 17
3.4 The Reinforcement Learning Model ......................... 21
    3.4.1 Expected Utility of the proposed match .......... 23
    3.4.2 Computing socially optimal matchings .......... 23
3.5 Reinforcement Learning Algorithm .......................... 25
    3.5.1 Epsilon Greedy ..................................... 25
    3.5.2 UCB1 .................................................. 27

**Chapter 4. Experiments** 28

4.1 Experimental setup ........................................ 28
4.2 Results ..................................................... 30
    4.2.1 Constant utilities ................................... 30
    4.2.2 Utilities from stationary distributions .......... 38
    4.2.3 Welfare for matchings with larger number of riders ... 46
Chapter 1

Introduction

1.1 Introduction

Ridesharing is a method of shared transportation where people share a car in order to reach their destination. Nowadays, people use ridesharing for most of their daily commute’s like travelling to office, grocery shopping, going to a party or to visit friends places. Each individual participating in the ridesharing platform can save money since it allows the co-travellers to divide the travel expenses. Conceptually, ridesharing is a system that can combine the flexibility and speed of private cars with the reduced cost of fixed-line systems [FDO+13].

The core of a ridesharing platform is to match riders with other suitable riders. A matching between two riders implies that the two riders share a single ride. A rider may prefer to share the ride with specific riders. Hence, a rider has preferences over the other riders. Based on some relevant criteria, the order that a rider provides to others is called the preference of that rider. This criterion can be social welfare between the riders. Social welfare is an essential aspect as it can determine the well-being of the riders [FS06]. The preferences of the riders can be represented as both cardinal and ordinal. The
preferences are depicted cardinally using utility functions [HGY18]. Different notions of social welfare can be used to provide matching between riders with high overall utility. The overall quality of the matching is measured by the social welfare of the matching. Existing studies focus on matching the riders based on social welfare [AD10, GZPH15, ACGD12, Les19]. The matching which guarantees the maximum social welfare is known as a socially optimal matching [FS06]. It provides the highest overall quality of the matching. Since it focuses only on the overall quality of the matching, it may not provide maximum benefit to each individual rider. A rider may believe there is a better match available than his current match. A matching is defined as a stable matching if no two riders, currently matched to others, would prefer to be matched together [Irv85]. Such a pair of riders is called a blocking pair to the matching. A stable matching guarantees individual satisfaction of the riders with their matches [WAE18]. The stability of a socially optimal matching can be computed by the number of blocking pairs in the matching [BMM10]. There are existing researches focused on providing stable matching between riders [PSJ+20, RC19, YLL+19, MYSJ19]. To provide a socially optimal matching, we need to know the preferences of the riders. There are situations where the utilities of the riders are not known beforehand. For instance, the riders participating in the ridesharing platform may not have any prior experience or exposure to sharing the rides with other riders. In those cases, we need to compute the socially optimal matching while learning the utility of the riders.

We design a model for computing the socially optimal matchings be-
tween the riders under different social welfare concepts while learning the utilities for the riders. We employ techniques from reinforcement learning [SB98] algorithms to determine the utilities of the riders based on their previous matchings. Upon receiving the expected utilities of the riders, we consider a variety of welfare measures to assess the quality of outcomes for a population of riders like utilitarian [Ben23], egalitarian [Har75], Nash [KN79] and, leximin social welfare [d’A85]. The matchings with maximum welfare for each measure are the socially optimal matchings. To check the stability of those matchings, we also provide a comparison between the number of blocking pairs in each socially optimal matching. We answer the following research questions in this thesis:

How do we learn the utilities of the riders to improve the social welfare among a population of riders in the ridesharing platforms? Which reinforcement learning algorithm performs better for learning the utilities of the riders? Which reinforcement learning algorithm performs better with respect to social welfare and stability?

1.2 Contributions

We introduce a multi-agent reinforcement learning model where each rider is modeled as a multi-armed bandit [Rob52]. It is similar to a multi-agent multi-armed bandit. The goal is to learn the expected utilities of the riders while maximizing their social welfare. We consider four well-established reinforcement learning algorithms: \( \epsilon \)-greedy [Wat89] with \( \epsilon = 0.0, \epsilon = 0.01, \)
\( \epsilon = 0.1 \), and UCB1 [ACBF02]. The \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \) outperforms all other reinforcement learning algorithms in terms of accuracy in learning expected utilities of the riders. The \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \) learns the expected utilities of the riders faster than UCB1 algorithm with better accuracy while UCB1 algorithm learns the expected utilities at a steady rate. The \( \epsilon \)-greedy algorithm with \( \epsilon = 0.0 \) gets stuck at a sub-optimal matching. Due to less exploration in the \( \epsilon \)-greedy algorithm with \( \epsilon = 0.01 \), it learns the expected utilities of the riders with lower accuracy than the \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \).

We provide socially optimal matchings using utilitarian, egalitarian, Nash, and leximin social welfare for expected utilities learned using each reinforcement algorithm and provide a comparison between them. UCB1 algorithm outperforms all other reinforcement learning algorithms to provide the maximum welfare in the socially optimal matchings.

Additionally, we also compute the number of strong and weak blocking pairs in each socially optimal matching to compare their stability. Compared to UCB1 algorithm, the \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \) has fewer number of strong blocking pairs in most of the socially optimal matchings. This indicates that socially optimal matchings computed using \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \) are more stable than socially optimal matchings computed using UCB1 algorithm.

This thesis is organised as follows. Chapter 2 talks about other significant researches in the field of matching, social welfare, and learning utilities
in the ridesharing platform. Chapter 3 provides model formulation along with basic terminology and definitions. It introduces the definition of matching, social welfare, and stability of socially optimal matchings. Also, the model formulation for the reinforcement learning algorithm is described along with the algorithms implemented. The experimental set-up and results of the model are described in Chapter 4. Finally, in Chapter 5, the conclusion is drawn, and the future directions of the thesis are listed out along with the advantages and disadvantages of the model.
Chapter 2

Related Works

A stable matching is defined as the matching where there are no two agents, who prefer each other rather than their current partner. The stable marriage problem devised by Gale and Shapley [GS62] confirm that between two equally sized sets of agents (men and women), there always exists a stable matching. A more general and algebraic context for the stable marriage problem has been developed by Gusfield and Irving [GI89]. A generalization to the stable matching problem is the stable roommate problem [Irv85]. As opposed to stable marriage problem where the matching is between two equally sized sets, the stable roommate problem provides stable matching for agents of the same even-sized set. For a stable roommate problem, there is no guarantee for a stable matching to exist. But, the stable roommate problem guarantees to find a stable matching, if one exists.

Learning of the user preferences by maintaining high user desirability has been proposed by Montazery and Wilson [MW16]. In their model they consider two disjoint sets of drivers $(D)$ and riders $(R)$ and all requests are defined by the set $S = R \cup D$. They define matching as a bipartite graph $G = (D, R, E)$, with $E$ representing the edges of the graph, and for $S_i, S_j \in S$, 
the gain of the matching \( S_i \) and \( S_j \) is the weight \( c_{ij} \). They propose the model to provide a set of best possible matching concerning the weight \( c_{ij} \). Since the model learns the preferences from a training data, this is a supervised learning model. They classify the supervised learning model as a regression problem where the model learns a scoring function from the preference list. The scoring function measures the expected satisfaction for the individual of the matching by generating a value for each matching [MW16]. To learn the scoring function, they develop a deduction of conventional Support Vector Machine (SVM) [Joa02]. The experiments are performed in real ridesharing records.

An online stable approach for preference-aware agents in online taxi dispatching has been proposed by Zhao et al. [ZXS+19]. They define their model as online stable matching under known identical independent distributions which maximizes the expected total profits and also try to satisfy the preferences among drivers and riders by minimizing the expected total number of blocking pairs [ZXS+19]. They define matching as a bipartite graph \( G = (U, V, E) \), where \( U \) represents the set of offline drivers, \( V \) represents set of online riders, and the edge \( f = (u, v) \) represents that driver \( u \) is matched with \( v \) which is also associated with a distance \( d(u, v) > 0 \). They define \( w_v > 0 \) as the profit for \( v \) when it is matched. They define preference over the sets such that if \( w_v > w_{v'} \), \( u \) prefers \( v \) over \( v' \), and if \( d(u, v) > d(u', v) \), \( v \) prefers \( u \) over \( u' \). The model designs an linear programming algorithm to provide,

1. \( \max \mathbb{E} \left[ \sum_{v \in V_M} w_v \right] \), where \( V_M \) is set of riders matched in \( M \)
2. \( \min \mathbb{E}[BP(M)] \), where \( BP(M) \) represented number of blocking pairs in \( M \) [ZXS^{+}19].

They also provide a competitive ratio to evaluate the performance of their online maximization algorithm.

Learning preferences of agents for matching is designed as a two-sided bandit problem for dating markets by Das and Kamenica [DK05]. Their model focuses on learning of agents in a one-to-one two-sided matching. According to their model, \( M \) is the set of men and \( W \) is set of women participating in the dating market for \( T \) time. They define matching similar to the definition of stable marriage problem [GS62]. The examine three matching mechanism: Gale Shapley matching [GS62], simultaneous offers, and sequential offers [DK05]. The model provides the matching by implementing \( \epsilon \)-greedy algorithm with different variations of \( \epsilon \).

The social welfare in one-sided matching markets for efficiently allocating \( n \) items to \( n \) agents who have preferences over the items is studied by Bhalgat et al. [BCK11]. They examine two natural measures of social welfare: ordinal social welfare and linear social welfare. According to their model, ordinal social welfare measures the satisfaction of each agent and linear welfare measures the decreasing agent’s utility over the preference lists. They analyze two matching mechanisms: random serial dictatorship [AS98] to allocate items fairly among the agents using randomly chosen permutation, and
a random assignment algorithm known as probabilistic serial (PS) mechanism [BM01, BCK11].
Chapter 3

Model

3.1 Model

The agents are the riders participating in the ridesharing model. It is defined as the set $N = \{1, 2, 3, \ldots, 2n\}$. Without loss of generality, we focus on even number of riders. So the total number of riders are $|N| = 2n$. Let the set of all matching be $M$.

A matching between the riders is defined as a one-to-one mapping between riders in the set $N$,

$$
\mu : N \rightarrow N
$$

such that $\mu(i) \neq \mu(j), \forall i, j \in N$. A rider can only be matched with one other rider. Currently, the model only supports two riders sharing a ride. Therefore, it is assumed that only two riders can share a ride.

Example 1. Assume there are four riders $r_1, r_2, r_3, r_4$, participating in the ridesharing algorithm. Therefore, $N = \{r_1, r_2, r_3, r_4\}$. An instance of the matching would be $\mu(r_1) = r_2$ and $\mu(r_3) = r_4$, i.e. $r_1$ is matched with $r_2$ and $r_3$ is matched with $r_4$. 

10
The satisfaction of the rider faced with options is denoted by preference of the rider [ASS02]. The preference profile represents the preference of the each rider \(i \in N\) over the set of alternate riders, \(N \setminus i\). We consider two kinds of preference profile, cardinal and ordinal.

**Utility** is an evaluation measure for cardinal preference profile [Fis68]. Each rider has a utility over all other riders, denoted as \(U_i\). The set of all utilities can be shown as a \(2n \times 2n\) matrix as follows,

\[
U = \begin{bmatrix}
-\infty & u_{1,2} & \ldots & u_{1,2n} \\
 u_{2,1} & -\infty & \ldots & u_{2,2n} \\
 \vdots & \vdots & \ddots & \vdots \\
 u_{2n,1} & u_{2n,2} & \ldots & -\infty 
\end{bmatrix}
= (u_{i,j}) \in [0,1]^{2n \times 2n}, \forall i, j \neq j
\]

where each element \(u_{i,j}\) represents the utility of \(\mu(i) = j\), i.e., the utility of rider \(i\) for rider \(j\). We assume that a rider cannot be matched with himself and no rider remains unmatched. Therefore we consider the minimal possible utility of a rider for himself, \(\forall i \in N, u_{i,i} = -\infty\). When the utilities are uncertain, we use expected utility. The expected mean utility of a rider is defined as the mean of the expected utilities of the matching between the rider and the alternate riders. For a rider \(i \in N\), the expected mean utility will be calculated as

\[
\frac{1}{2n-1} \sum_{k \in N \setminus i} u_{i,k}
\]

In Example 1, assume the expected utilities of rider \(r_1\) when sharing the ride with \(r_2\) is 0.8, with \(r_3\) is 0.6 and with \(r_4\) is 0.9. So, the expected utility vector for \(r_1\) will be \(U_{r_1} = [-\infty, 0.8, 0.6, 0.9]\). The expected mean utility of
rider $r_1$ would be $\frac{0.8+0.6+0.9}{3} = 0.77$. Similarly, assume the utility vectors for riders $r_2, r_3, r_4$ are $U_{r_2} = [0.6, -\infty, 0.7, 0.8]$, $U_{r_3} = [0.2, 0.7, -\infty, 0.5]$, and $U_{r_4} = [0.8, 0.2, 0.1, -\infty]$, respectively.

![Figure 3.1: Depiction of Example 1 where the possible matchings of riders are represented by the edges and each weight on an edge represents the expected utility of rider $r_i$ for rider $r_j$ ($u_{i,j}$), where $i, j \in [1, 2, 3, 4]$.](image)

### 3.2 Social Welfare Functions

Social welfare is a measure to determine the well-being of the riders. The accumulation of the preference of the riders can be defined using the concept of social welfare [FS06]. Various aspects of social welfare measure can be utilized to provide matching between riders with high overall quality. A matching which provides maximum social welfare is called a socially optimal matching. Socially optimal matching guarantees the highest overall quality of
the matching. To investigate the performance of the algorithm, we consider various social welfare concepts to provide socially optimal matchings. The matching procedure is centralized, meaning the algorithm will decide the final matching for the riders after considering the expected utilities of the riders.

3.2.1 Utilitarian Social Welfare

Utilitarian social welfare [Ben23] is defined as the measure of the quality of the matching from the outlook of the system as a whole [CDU+06]. It can be calculated by summing the expected utilities of the riders for a matching.

\[ sw_{util}(\mu) = \sum_{i \in N} u_{i,\mu(i)} \]

The socially optimal matching based on maximum utilitarian social welfare is defined as,

\[ \argmax_{\mu \in M} sw_{util}(\mu) \]

In Example 1, the expected utilities are,

\[ U = \begin{bmatrix} -\infty & 0.8 & 0.6 & 0.9 \\ 0.6 & -\infty & 0.7 & 0.8 \\ 0.2 & 0.7 & -\infty & 0.5 \\ 0.8 & 0.3 & 1 & -\infty \end{bmatrix} \]

The set of all possible matchings \( M \) would be \( \mu_1 = [(r_1, r_2), (r_3, r_4)] \), \( \mu_2 = [(r_1, r_3), (r_2, r_4)] \), and \( \mu_3 = [(r_1, r_4), (r_2, r_3)] \).

The utilitarian social welfare of the matchings are,

\[ sw_{util}(\mu_1) = 0.8 + 0.6 + 0.5 + 1 = 2.9 \]
Figure 3.2: The expected utilities of the riders in matching $\mu_1$, $\mu_2$, and $\mu_3$, respectively. The circles indicate the expected utilities of the matched riders.

$$sw_{util}(\mu_2) = 0.6 + 0.8 + 0.2 + 0.2 = 1.8$$

$$sw_{util}(\mu_3) = 0.9 + 0.7 + 0.7 + 0.8 = 3.1$$

Hence, the socially optimal matching based on utilitarian social welfare is the one with the maximum utilitarian social welfare which is $\mu_3 = [(r_1, r_4), (r_2, r_3)]$.

### 3.2.2 Egalitarian Social Welfare

Egalitarian social welfare [Har75] is defined as the expected utility of $\mu(i) \in \mu$ for $i \in N$ that is worst off in matching $\mu$ [CDU+06].

$$sw_{egal}(\mu) = \min_{i \in N} u_{i, \mu(i)}$$

The socially optimal matching based on maximum egalitarian social welfare is defined as,

$$\arg\max_{\mu \in M} sw_{egal}(\mu)$$

In Example 1, the egalitarian social welfare of the matchings are,

$$sw_{egal}(\mu_1) = 0.5$$
\[ sw_{egal}(\mu_2) = 0.2 \]
\[ sw_{egal}(\mu_3) = 0.7 \]

Hence, the socially optimal matching based on egalitarian social welfare is the one with the maximum egalitarian social welfare which is \( \mu_3 = [(r_1, r_4), (r_2, r_3)] \). In this case, the socially optimal matching using both utilitarian and egalitarian social welfare is same. However, this does not indicate the socially optimal matching for utilitarian and egalitarian social welfare will remain same for all cases.

### 3.2.3 Nash Social Welfare

The Nash social welfare [KN79] is defined as the product of the expected utilities of the riders, given the matching [BCH+17].

\[ sw_{Nash}(\mu) = \prod_{i \in N} u_{i, \mu(i)} \]

The socially optimal matching based on Nash product is defined as,

\[ \arg\max_{\mu \in M} sw_{Nash}(\mu) \]

In Example 1, the Nash social welfare of the matchings are,

\[ sw_{Nash}(\mu_1) = 0.8 \times 0.6 \times 0.5 \times 1 = 0.24 \]
\[ sw_{Nash}(\mu_2) = 0.6 \times 0.8 \times 0.2 \times 0.2 = 0.0192 \]
\[ sw_{Nash}(\mu_3) = 0.9 \times 0.7 \times 0.7 \times 0.8 = 0.3528 \]

Hence, the socially optimal matching based on Nash social welfare is the one with the maximum Nash product which is \( \mu_3 = [(r_1, r_4), (r_2, r_3)] \).
3.2.4 Leximin Social Welfare

The leximin social welfare [d’A85] is a social welfare ordering that refines egalitarian social welfare [CDU+06]. It works same as egalitarian social welfare by comparing the minimum utilities of each matching settings $\mu \in M$. It is different from egalitarian social welfare as when the minimum utilities coincide, it compares the utilities of the next matching with minimum utilities in the coinciding matching settings, and so on until it finds a utility which is higher than the utility of the other matching. It is denoted by $sw_{lex}$.

The socially optimal matching based on leximin ordering is defined as,

$$\arg\max_{\mu \in M} sw_{lex}(\mu)$$

In Example 1, the leximin social welfare of the matchings are,

$$sw_{lex}(\mu_1) = 0.5$$
$$sw_{lex}(\mu_2) = 0.2$$
$$sw_{lex}(\mu_3) = 0.7$$

Hence, the socially optimal matching based on leximin ordering is $\mu_3 = [(r_1, r_4), (r_2, r_3)]$.

Example 2. There are four riders $r_1, r_2, r_3,$ and $r_4$. Assume the utility matrix of the riders is

$$U = \begin{bmatrix} -\infty & 0.6 & 0.7 & 0.4 \\ 0.4 & -\infty & 0.9 & 0.6 \\ 0.7 & 0.6 & -\infty & 0.8 \\ 0.7 & 0.5 & 0.6 & -\infty \end{bmatrix}$$
The set of matchings \( M \) is \( \mu_1 = [(r_1, r_2), (r_3, r_4)], \) \( \mu_2 = [(r_1, r_3), (r_2, r_4)] \), and \( \mu_3 = [(r_1, r_4), (r_2, r_3)] \). The minimum utility of the matchings are 0.4 for \( \mu_1 \), 0.4 for \( \mu_2 \), 0.6 for \( \mu_3 \), and 0.5 for \( \mu_4 \). Since the minimum utility of matching in \( \mu_1 \) and \( \mu_2 \) are equal, the next minimum utility is computed which are 0.6 for \( \mu_1 \) and 0.6 for \( \mu_2 \) which are equal again. The next minimum utility are 0.7 for \( \mu_1 \) and 0.9 for \( \mu_2 \). Since \( \mu_2 \) has the higher utility, the socially optimal matching based on leximin ordering is \( \mu_2 = [(r_1, r_3), (r_2, r_4)] \).

### 3.3 Stable Matching

Stability is a desirable property of matching as there is empirical evidence that links stability to market failures [Rot00]. According to the stable roommate problem [Irv85], a matching is stable if there are no two riders, who prefer each other rather than their matches. Such a pair is said to be a blocking pair with respect to the matching. For an even-sized set, a stable matching can be found by the the stable roommate problem. For certain number of riders and their preferences, a stable matching may fail to exist. Stable roommate problem guarantees to find a stable matching, if it exists. Stable matching guarantees that no two riders will form a blocking pair. A matching with fewer number of blocking pairs is considered to be more stable than a matching with higher number of blocking pairs.

**Strong Blocking Pair.** A pair \((i, j)\), where \(i, j \in N\) is in matching \(\mu\), is called a strong blocking pair, if both \(i\) and \(j\) strictly prefer each other more
than their matched rider \(\mu(i)\) and \(\mu(j)\). Pair \((i, j)\) is a strong blocking pair if

\[
(i, j) \text{ for } \forall i, j \in N, \ u_{i,j} > u_{i,\mu(i)} \text{ and } u_{j,i} > u_{j,\mu(j)}
\]

A matching with strong blocking pair \((i, j)\) suggests that both riders in pair \((i, j)\) are not satisfied with their match.

**Weak Blocking Pair.** A pair \((i, j)\), where \(i, j \in N\) is in matching \(\mu\), is called a weak blocking pair, if both \(i\) and \(j\) prefer each other at least as much as their matched rider \(\mu(i)\) and \(\mu(j)\). Pair \((i, j)\) is a weak blocking pair if

\[
(i, j) \text{ for } \forall i, j \in N, \ u_{i,j} \geq u_{i,\mu(i)} \text{ and } u_{j,i} > u_{j,\mu(j)}
\]

A matching with weak blocking pair \((i, j)\) suggests that the rider \(i\) is as much satisfied with rider \(j\) as with its matched rider but rider \(j\) strictly prefers \(i\) more than its matched rider.

The strong blocking pairs are subsets of weak blocking pairs. All strong blocking pairs are weak blocking pairs but not vice-versa. A matching \(\mu\) is called a weakly stable matching if there is no strong blocking pair in the matching \(\mu\). If there is no weak blocking pair in matching \(\mu\), then the matching is called a strongly stable matching.

In Example 1, the socially optimal matching based on utilitarian, egalitarian and Nash social welfare is \(\mu_3 = [(r_1, r_4), (r_2, r_3)]\). For the pairs \((r_1, r_2)\) and \((r_1, r_3)\), both prefer their matching than each other. For the pair \((r_2, r_4)\), \(u_{r_2,r_4} > u_{r_2,\mu_3(r_2)}\) but \(u_{r_4,\mu_3(r_4)} > u_{r_4,r_2}\). There are no two riders in the matching
\( \mu_3 \) who strictly prefer each other than their matches. There are no two riders in the matching \( \mu_3 \) who prefer each other at least as much as their matches. So there are no strong or weak blocking pairs in the matching \( \mu_3 \), so \( \mu_3 \) is a strongly stable matching. The socially optimal matching based on leximin ordering is \( \mu_2 = [(r_1, r_3), (r_2, r_4)] \). For pair \( (r_1, r_4) \), \( u_{r_1, r_4} > u_{r_1, \mu(r_1)} \) and \( u_{r_4, r_1} > u_{r_4, \mu(r_3)} \), which is a strong blocking pair. Hence, \( \mu_2 \) is not a stable matching.

Example 3. There are four riders \( r_1, r_2, r_3, \) and \( r_4 \). Assume the expected utilities of the riders are,

\[
U = \begin{bmatrix}
-\infty & 0.6 & 0.7 & 0.5 \\
0.5 & -\infty & 0.9 & 0.6 \\
0.7 & 0.6 & -\infty & 0.5 \\
0.5 & 0.4 & 0.6 & -\infty
\end{bmatrix}
\]

The set of matchings \( M \) is \( \mu_1 = [(r_1, r_2), (r_3, r_4)] \), \( \mu_2 = [(r_1, r_3), (r_2, r_4)] \), and \( \mu_3 = [(r_1, r_4), (r_2, r_3)] \).

\[ sw_{util}(\mu_1) = 0.6 + 0.5 + 0.5 + 0.6 = 2.2 \]
\[ sw_{util}(\mu_2) = 0.7 + 0.7 + 0.6 + 0.4 = 2.4 \]
\[ sw_{util}(\mu_3) = 0.5 + 0.5 + 0.9 + 0.6 = 2.5 \]

The socially optimal matching based on utilitarian social welfare is \( \mu_3 = [(r_1, r_4), (r_2, r_3)] \). For pair \( (r_1, r_3) \), \( u_{r_1, r_3} > u_{r_1, \mu(r_1)} \) and \( u_{r_3, r_1} > u_{r_3, \mu(r_3)} \), which is a strong blocking pair. Hence, \( \mu_3 \) is not a stable matching (Figure 3.3).

The Nash social welfare of the set of matchings \( M \) are,

\[ sw_{Nash}(\mu_1) = 0.6 \times 0.5 \times 0.5 \times 0.6 = 0.09 \]
The socially optimal matching based on Nash social welfare is \( \mu_3 = [(r_1, r_4), (r_2, r_3)] \), which is not a stable matching.

The egalitarian social welfare of the set of matchings \( M \) are,

\[
sw_{egal}(\mu_1) = 0.5 \\
sw_{egal}(\mu_2) = 0.4 \\
sw_{egal}(\mu_3) = 0.5
\]

The socially optimal matching based on egalitarian social welfare is \( \mu_2 = [(r_1, r_3), (r_2, r_4)] \). There are no two riders in the matching \( \mu_2 \) who strictly prefer each other than their matches. There are no two riders in the matching \( \mu_2 \) who prefer each other at least as much as their matches. Since, there are no strong or weak blocking pairs in the matching \( \mu_2 \), \( \mu_2 \) is a strongly stable matching (Figure 3.4).
\[ U = \begin{bmatrix} -\infty & 0.6 & 0.7 & 0.5 \\ 0.5 & -\infty & 0.9 & 0.6 \\ 0.7 & 0.6 & -\infty & 0.5 \\ 0.5 & 0.4 & 0.6 & -\infty \end{bmatrix} \]

Figure 3.4: Stable matching \( \mu_2 \) where utilities of socially optimal matching using egalitarian social welfare is circled

### 3.4 The Reinforcement Learning Model

The reinforcement learning model is based on a similar setting of stochastic \( n \)-armed bandit [Rob52, SB98]. In the multi-armed bandit problem, a bandit has \( n \) different options to choose from, where \( n \) is the number of arms for the bandit [SB98]. After each action, it receives a reward chosen from some probability distribution. The goal of the multi-arm bandit problem is to maximize the total reward received over the time period.

In our model, each rider \( i \in N \) is modeled as a bandit, and the alternate riders \( N \setminus i \) are modeled as arms of the bandit. In Example 1, for rider \( r_1 \) as bandit, the arms will be \( r_2, r_3, \) and \( r_4 \). Similarly, for rider \( r_2 \) as bandit, the arms are \( r_1, r_3, \) and \( r_4 \). For rider \( r_3 \) as bandit, the arms are \( r_1, r_2, \) and \( r_4 \), and for rider \( r_4 \) as bandit, the arms are \( r_1, r_2, \) and \( r_3 \). There are \( 2n \) multi-armed bandit problems running simultaneously for learning the respective expected utilities while maximizing their social welfare. The model is similar to a multi-agent multi-armed bandit. Our reinforcement learning model is a tuple \( < N, U, M > \)
where $N$ is the set of riders, $U$ denotes the expected utilities of the riders, and $M$ denotes the set of all matchings.

**Action.** The set of actions are denoted as the set of all possible matchings $M$.

**Reward.** The individual reward of a rider is denoted as the expected utility $u_{i,j}, \forall i, j \in N$. The total reward for all the riders is the expected utility matrix $U$.

For each rider, $i \in N$, the model has $2n - 1$ possible matches to choose from, and there are $|T|$ time steps. Let $\mu^t(i)$ be the proposed match for rider $i \in N$, at time step $t \in T$. The model’s goal is to maximize the expected mean utility of each rider $i$ over $T$ time steps. As in the case of stochastic bandits [Sli19], the three essential assumptions in the model are:

- The model observes nothing else but only the utility for the proposed match.

- For each rider $i \in N$, there is a distribution $\mathcal{U}_{i,j}$ for each alternate rider $j \in N$ called the utility distribution. For each proposed match, the expected utility is sampled independently from this distribution, which is initially unknown to the model. The expected utilities are bounded to interval $[0, 1]$ for simplicity.
• The utility distributions are fully specified by the expected utilities of the respective distributions. For each rider \( i, j \in N \), the expected utility of the distribution \( U_{i,j} \) is,

\[
    u_{i,j} = E[U_{i,j}]
\]

3.4.1 Expected Utility of the proposed match

At each time step, for each rider, utility is received as a reward for a proposed match. Given a matching, the expected utility is the average of the utilities of the riders for their respective matches. The problem with averaging the utilities is that its memory and computational requirements grow over time without bound [SB98]. Therefore, an incremental update formula [SB98] is devised for computing the averages. For rider \( i \in N \), let \( u_{i,\mu(i)}^k \) be the utility of rider \( i \) received when matched with \( \mu(i) \) for the \( k^{th} \) time. Initially the expected utility is defined by a default value, such as \( u_{i,\mu(i)} = 0 \). The incremental update formula for computing the expected utility of the rider \( i \) is

\[
    \bar{u}_{i,\mu(i)} = \bar{u}_{i,\mu(i)} + \frac{1}{k} [u_{i,\mu(i)}^k - \bar{u}_{i,\mu(i)}]
\]

3.4.2 Computing socially optimal matchings

At each time step, for each rider, the algorithm learns the expected utility of the proposed match for the rider and updates the expected utility matrix \( U \). Once the algorithm has proposed matches for all riders for the time step, using the updated expected utility matrix \( U \), it computes the social
welfare based on utilitarian, egalitarian, Nash and leximin ordering (as described in Section 3.2) for all possible matchings \( \mu \in M \). The matchings with the maximum social welfare for each welfare concept are the socially optimal matchings for the time step.

Constant utilities and Utilities from stationary distributions. *Constant utilities* are the utilities that are fixed and do not change over time. For each rider \( i \in N \), \( 2n - 1 \) utilities are sampled from the respective distributions and they remain the same over the time-steps. The utilities that are sampled from the respective stationary distributions at each time step are called *utilities from stationary distribution*. For each time step and for each rider \( i \in N \), \( 2n - 1 \) utilities are sampled from the respective distributions.

For each rider \( i \in N \), the best proposed match, *i.e.*, the match with maximum utility in the utility vector \( U_i \) is denoted by,

\[
u^*_i = \max_{k \in N} u_{i,k}\]

The performance measure of the model is evaluated by minimizing the *total regret* [SB98] for each rider which is formally defined as,

\[
\forall i \in N, \quad R_i(T) = u^*_i \cdot T - \sum_{t=1}^{T} u_{i,\mu^t(i)}
\]

Note that \( \mu^t(i) \) is a random quantity as it depends on the utilities sampled from the distribution at the time step \( t \), so \( R_i(T) \) is also a random variable. The expected regret of the rider \( i \) is denoted by \( \mathbb{E}[R_i(T)] \).
At each time step, the reinforcement learning algorithm is faced with choice of whether to use to current information it has gained to propose a match, or try to learn more and propose a random match. The fundamental choice of reinforcement learning models is the exploration and exploitation trade-off [Mar91]. Exploration indicates searching for better choices and exploitation refers to make the best choice based on the current knowledge.

3.5 Reinforcement Learning Algorithm
3.5.1 Epsilon Greedy

A greedy algorithm always takes the action that has the best payoff at that moment. This might lead to bad long term consequences. Similar to greedy algorithm, the epsilon greedy or $\epsilon$-greedy algorithm [Wat89] takes the action with the best payoff, however from time to time it explores the other available actions [Whi12]. It is widely used because it is very simple, and is very successful in most empirical problems. It provides a mix of exploration and exploitation strategies.

The $\epsilon$-greedy algorithm selects random proposed match with $\epsilon$ probability and best proposed match with $(1 - \epsilon)$ probability. A random proposed match is a match proposed uniformly at random from the set of all matchings $M$. The proposed match $\mu(i)$ for rider $i \in N$, given the utility $u_{i,\mu(i)}$, is selected as,

$$
\begin{cases}
\arg\max_{\mu \in M} u_{i,\mu(i)}, & \text{if probability } 1 - \epsilon \\
\text{random proposed match}, & \text{if probability } \epsilon
\end{cases}
$$
For example, if $\epsilon = 0.05$, the algorithm will exploit the best option 95% of the time and will explore random alternatives 5% of the time. The constant exploration probability $\epsilon$ provides linear growth in total expected regret. According to Kuleshov and Precup [KP14], for each time step $t$ and for each rider $i \in N$, $\epsilon$-greedy algorithm with exploration probability $\epsilon = t^{-\frac{1}{3}} \cdot (|N| \log t)^{1/3}$ achieves regret bound,

$$\mathbb{E}[R_i(t)] \leq t^{2/3} \cdot O(|N| \log t)^{1/3}$$

The above equation indicates the regret for learning the expected utilities of the riders in our algorithm. Since exploration is random as it chooses the match uniformly at random from the set of matchings $M$, there is no loss of expectation. The $\epsilon$-greedy algorithm guarantees the expected mean utility will gradually converge towards the utility sampled from the distribution for all riders.

**Example 4.** Assume there are four riders $r_1, r_2, r_3$ and $r_4$. For $r_1$, at first time step, the utilities for matching with $r_2, r_3,$ and $r_4$ is randomly sampled from probability distributions unknown to the mode as $[0.4, 0.4, 0.2]$. Let $\epsilon = 0.1$, initial expected utility vector for $r_1$ is $[-\infty, 0, 0, 0]$, and expected mean utility of $r_1$ is 0. For first time step, the probability is 0.02. So $r_1$ will choose a match randomly between the alternative riders. Say, it chooses $r_3$, then $\mu(r_1) = r_3$ and $u_{r_1,r_3} = 0.4$. Now, estimated utility vector for $r_1$ is $[-\infty, 0, 0.4, 0]$ and mean estimated utility of $r_1$ is 0.4. For second time step, the probability is now 0.3, so $r_1$ will choose a match with maximum expected
utility, which is $\mu(r_1) = r_3$. After every time step is completed, the expected utility vector and expected mean utility for $r_1$ is saved and reinitialized for the next time step with another sample from the probability distributions.

### 3.5.2 UCB1

The UCB1 algorithm is part of the UCB family of algorithms as a simpler algorithm based on principle of optimism during uncertainty \cite{ACBF02}. The UCB1 algorithm builds an optimistic guess on the best proposed match and chooses that action. If the guess is wrong, the payoff for the optimistic guess decreases and the optimistic guess changes to another action. If the guess is right, it exploits the actions. This results in minimal regret.

UCB1 maintain the number of times that each match has been proposed for each rider $i \in N$, denoted by $m_{i,\mu(i)}(t)$, in addition to the expected mean utilities. Initially each match is selected once. Then, at time-step $t \in |T|$, the algorithm selects the match $\mu^t(i)$ as

$$\forall i \in N, \mu^t(i) = \arg\max_{\mu \in M} \left( u_{i,\mu(i)} + \sqrt{\frac{2 \ln t}{m_{i,\mu(i)}}} \right)$$

According to Auer et.al. \cite{ACBF02}, at time-step $t$, for rider $i$, the expected regret of UCB1 is bounded by

$$\mathbb{E}[R_i(t)] \leq 8 \times \sum_{i: u_{i,\mu(i)} < u^*_i} \frac{\ln t}{u^*_i - u_{i,\mu(i)}} + (1 + \frac{\pi^2}{3}) \sum_{\mu \in M} (u^*_i - u_{i,\mu(i)})$$

UCB1 is bounded by $O(\log t)$ regret in learning the expected utilities of the riders.

27
Chapter 4

Experiments

4.1 Experimental setup

The model is implemented in Python. The experiments are run for up to 14 riders. For each experiment, the utilities are sampled from either of the three distributions: truncated normal distribution, uniform distribution, and binomial distribution. The distributions are selected for comparing the performance of the algorithm on different utility samples. The experiment is implemented with three values of $\epsilon$ for $\epsilon$-greedy algorithm: $\epsilon = 0$, $\epsilon = 0.01$, $\epsilon = 0.1$, and UCB1 algorithm. Each experiment is run for 1000 simulations for constant utilities and for utilities from stationary distributions. The code snippets for the algorithms are provided in Appendix B. Below we describe each of the distributions.

**Truncated Normal Distribution.** The truncated normal distribution is a continuous probability distribution derived from normal distribution which is made finite by bounding either below or above or both [Bur14]. For this experiment, the distribution is bounded between 0 and 1. Each distribution has a different mean and variance, which are uniformly sampled between 0 and 1.
**Uniform Distribution.** The uniform distribution is a continuous probability distribution that lies between certain bounds where each sample of the distribution has the same probability [DKLM05]. The bounds are defined by the minimum and maximum values, which are 0 and 1, respectively.

**Binomial Distribution.** The binomial distribution is a discrete probability distribution where the values are based on number of successes based on the success probability $p$: 1 (with probability $p$) and 0 (with probability $1 - p$) [Wyp14]. For this experiment, the value of $p$ is sampled uniformly at random from values ranging from 0.1 to 0.9.

The steps involved in each experiment are as follows:

1. Sample utilities from respective distributions.

2. Run the reinforcement learning algorithms for all the riders. The algorithm does the following:
   
   (a) Learn the expected utilities of all riders while maximizing the social welfare of the riders.

   (b) Compute maximum welfare to provide the socially optimal matching using the following:
      
      i. Utilitarian social welfare
      
      ii. Egalitarian social welfare
      
      iii. Nash social Welfare
iv. Leximin social welfare

(c) Compute number of strong and weak blocking pairs in each socially optimal matching.

4.2 Results

The results are displayed for the algorithm running on 14 riders. The results of the algorithms running on 6 and 10 riders are provided in Appendix A.

4.2.1 Constant utilities

Constant utilities are the utilities that are fixed over time. For each rider $i \in N$, $2n - 1$ utilities are sampled from the respective distributions and used for all the time-steps.

**Learning expected utilities.** Figure 4.1 compares the three $\epsilon$-greedy algorithms ($\epsilon = 0.0, 0.01, 0.1$) and UCB1 algorithm for 14 riders on constant utilities from truncated normal distributions. The $\epsilon$-greedy algorithm with $\epsilon = 0.0$ levels off at a lower level because it often gets stuck selecting sub-optimal proposed match due to no exploration. The $\epsilon$-greedy algorithm with $\epsilon = 0.01$ and $\epsilon = 0.1$ eventually perform better because they continue to explore and to improve the chances of recognizing the best-proposed match. The $\epsilon$-greedy algorithm with $\epsilon = 0.1$ explores more and finds the best proposed match earlier. UCB1 performs at a more steady rate by calculating the
Figure 4.1: Percentage of 14 riders learning their best match with constant utilities from truncated normal distributions.

optimistic guess of the best proposed match at each time step. The initial spike in UCB1 accuracy is due to selecting each proposed match once during the initial time-steps. The result is consistent with smaller number of riders.

**Welfare of socially optimal matchings.** For computing Nash welfare of a matching, we need to find the product of the utilities of each rider. Since, utility of a rider could be zero, the resulting Nash welfare of the matching also becomes zero. Therefore, we compute the Nash welfare of a matching by taking the product of only non-zero utilities of the riders in a matching.

31
Figure 4.2 shows the welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering from the expected utilities learned by the three \( \epsilon \)-greedy and UCB1 algorithms for 14 riders with constant utilities from truncated normal distributions. The \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \) and UCB1 algorithms provide the maximum social welfare in each of the socially optimal matchings. The \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \) converges earlier than UCB1 algorithm. The \( \epsilon \)-greedy algorithm with \( \epsilon = 0.1 \) converges in about 100 time-steps while UCB1 algorithm takes about 350 time-steps to converge. Both \( \epsilon \)-greedy with \( \epsilon = 0.1 \) and UCB1 converges to the actual welfare. The results of the \( \epsilon \)-greedy algorithm with \( \epsilon = 0.0, \epsilon = 0.1 \), and UCB1 algorithm holds for smaller number of riders (Figure 4.3). However, the \( \epsilon \)-greedy algorithm with \( \epsilon = 0.01 \) converges faster as the number of riders decrease. The jumps in the plots of Figure 4.3 indicates exploration by \( \epsilon \)-greedy with \( \epsilon = 0.1 \) where they explore other other matches rather than pick the match with maximum utility for the time step.
Figure 4.2: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 14 riders with constant utilities from truncated normal distributions (Legend same for all image).
Figure 4.3: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 6, 8, and 10 riders with constant utilities from truncated normal distributions (Legend same for all image).
**Blocking pairs in socially optimal matchings.** For constant utilities in general, the $\epsilon$-greedy algorithm with $\epsilon = 0.1$ outperforms the other three algorithms in terms of accuracy in learning the expected utilities of the riders. However, UCB1 algorithm provides higher social welfare in the socially optimal matchings.

All strong blocking pairs are likewise weak blocking pairs, so the strong blocking pairs are not counted in the plots of weak blocking pairs. Figure 4.4 and Figure 4.5 display the number of blocking pairs in the socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering from expected utilities learned by the $\epsilon$-greedy algorithm with $\epsilon = 0.1$ and UCB1 algorithms for 14 riders with constant utilities from truncated normal distributions. For some time steps, the number of strong and weak blocking pairs remain the same over the time steps because the reinforcement algorithms select the same proposed match over the time steps. Due to more exploration and selecting different proposed match at the initial time-steps, the strong and weak number of blocking pairs fluctuate for the $\epsilon$-greedy algorithm with $\epsilon = 0.1$. For UCB1 algorithm, the number of strong and weak number of blocking pairs is consistent over the time steps. However, for smaller number of riders, the number of strong and weak blocking pairs are consistent for both UCB1 and $\epsilon$-greedy algorithm with $\epsilon = 0.1$. For both $\epsilon$-greedy algorithm with $\epsilon = 0.1$ and UCB1, the socially optimal matching computed using utilitarian social welfare has fewer number of blocking pairs compared to that of egalitarian, Nash, and leximin social welfare. The result holds for smaller number of riders.
Figure 4.4: Number of blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 14 riders with constant utilities from truncated normal distributions.
Figure 4.5: Number of blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 14 riders with constant utilities from truncated normal distributions.
4.2.2 Utilities from stationary distributions

Utilities from stationary distributions are the utilities that are sampled for each time step. For each time-step and each rider $i \in N$, $2n - 1$ utilities are sampled from the respective distributions.

Learning expected utilities. Figure 4.6 compares the three $\epsilon$-greedy algorithms ($\epsilon = 0.0, 0.01, 0.1$) and UCB1 algorithm along with the error bars for 14 riders on utilities from stationary distributions using truncated normal distributions. The plots in the figure show the percentage of the best proposed match selected for the current time step. It does not indicate the best expected match for the riders. The reinforcement learning algorithms for utilities don’t usually converge since the utilities sampled are changing over each time-step, and there is no fixed value for the algorithms to converge. However, the greedy algorithm ($\epsilon = 0.0$) seems to perform as much as the other algorithms. The $\epsilon$-greedy algorithm with $\epsilon = 0.1$ appears to perform slightly better by exploring the choices more to find the best-proposed match. While the average percentage of best proposed match selected by the riders increase for all reinforcement algorithms with smaller number of riders, the result still holds.
Figure 4.6: Percentage of 14 riders learning their best match with utilities from stationary distributions using truncated normal distributions.
Welfare of socially optimal matchings. Figure 4.7 shows the welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering from the expected utilities learned by the three $\epsilon$-greedy and UCB1 algorithms for 14 riders with utilities from stationary distributions using truncated normal distributions. UCB1 algorithm provides the maximum utilitarian, egalitarian and leximin social welfare, which is slightly better than the $\epsilon$-greedy algorithm with $\epsilon = 0.1$ and $\epsilon = 0.01$. The maximum Nash social welfare is similar to all the algorithms. The initial spike in UCB1 algorithm is due to each rider choosing each possible match at least once to calculate the optimistic guess for finding the best rider to match with. Due to the high welfare during the initial time-step, the plot displays welfare as converging to zero over the time steps. However, the welfare for each algorithm is not zero but ranging between $0.00001$ to $0.2$. However, UCB1 provides slightly higher Nash social welfare. Also, UCB1 algorithm provides social welfare closest to the actual welfare of the matchings if the utilities were known. The result holds for smaller number of riders (Figure 4.8).
Figure 4.7: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 14 riders with utilities from stationary distributions using truncated normal distributions (Legend same for all image).
Figure 4.8: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 6, 8, and 10 riders with utilities from stationary distributions using truncated normal distributions (Legend same for all image).
Blocking pairs in socially optimal matchings. Figure 4.9 and Figure 4.10 display the number of blocking pairs in the socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering from expected utilities learned by $\epsilon = 0.1$ and UCB1 algorithms for 14 riders with utilities from stationary distributions using truncated normal distributions. $\epsilon = 0.1$ has less number of strong and weak blocking pairs in socially optimal matching using utilitarian social welfare than that of UCB1 algorithm. For both $\epsilon$-greedy algorithm with $\epsilon = 0.1$ and UCB1, the socially optimal matching computed using utilitarian social welfare has fewer number of blocking pairs compared to that of egalitarian, Nash, and leximin social welfare. The result holds for smaller number of riders. For both $\epsilon$-greedy algorithm with $\epsilon = 0.1$ and UCB1, the socially optimal matching computed using utilitarian social welfare has fewer number of blocking pairs compared to that of egalitarian, Nash, and leximin social welfare. The result holds for smaller number of riders.

The experiments with binomial and uniform distributions provide similar results to that of truncated normal distributions. The plots are shown in the Appendix A along with the individual accuracy plots with error bars for each $\epsilon$-greedy and UCB1 algorithm.
Figure 4.9: Number of blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 14 riders with utilities from stationary distributions using truncated normal distributions.
Figure 4.10: Number of blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 14 riders with utilities from stationary distributions using truncated normal distributions.
4.2.3 Welfare for matchings with larger number of riders

To provide the socially optimal matchings, we compute the social welfare of all possible matchings $M$ using utilitarian, egalitarian, Nash, and lexicimin ordering based on the expected utility matrix $U$ learned using the reinforcement learning algorithms. However, computing the social welfare for all possible matchings, in general, is an NP-hard problem [RR10]. The number of possible matchings increases exponentially as the number of riders increase. Figure 4.11 and Figure 4.12 show the welfare of matchings received from the reinforcement learning algorithms while learning the expected utilities at each time step for 50 and 100 riders, respectively. The utilities are sampled from stationary distributions using truncated normal distributions. The matchings are selected using a greedy approach. The expected utility of each match is computed based on the expected utility matrix learned. The matching is selected by the algorithm as follows:

\begin{algorithm}
\caption{Greedy approach to select matching for larger number of riders}
\begin{algorithmic}
\State \textbf{Input:} Utility matrix $U$
\State \textbf{Output:} Matching $\mu$
\State select match $(i,j)$ with highest utility and add to $\mu$;
\While{all riders not matched}
\State select next match $(i,j)$ with highest utility;
\If{riders $i,j$ not in $\mu$}
\State add to $\mu$;
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

We are not able to provide any insight on the socially optimal matchings for 50 and 100 riders since computing all the possible matchings for 50
and 100 riders are not possible with the current computational power. However, from the figures, we can infer the UCB1 algorithm outperforms all other reinforcement learning algorithms to provide the matchings with higher social welfare computed using utilitarian and Nash welfare concepts. The initial spike in the UCB1 algorithm is due to each rider choosing each possible match at least once. Due to the high welfare during the initial time-step, the plot displays welfare as converging to zero over the time steps. However, the welfare for each algorithm is not zero but ranging between $0.0000001 - 0.0001$. For welfare computed using egalitarian social welfare, the $\epsilon$-greedy algorithm with $\epsilon = 0.1$ provides higher welfare than the UCB1 algorithm in some time-steps, however, UCB1 algorithm provides consistent egalitarian social welfare of the matchings after the initial time steps. This happens mainly due to $\epsilon$-greedy algorithm with $\epsilon = 0.1$ exploring matches which has not been explored before (matches with zero utility), increasing the minimum utility of the matchings. The welfare computed using leximin social welfare is not shown as it is the same as the plot of egalitarian social welfare. Figure 4.13 and Figure 4.14 show the number of blocking pairs in the matchings selected using greedy approach for 50 and 100 riders, respectively. The $\epsilon$-greedy algorithm with $\epsilon = 0.1$ produce fewer number of blocking pairs in the matchings using greedy approach than the UCB1 algorithm.
Figure 4.11: The welfare of matchings based on greedy approach for 50 riders with utilities from stationary distributions using truncated normal distributions (Legend same for all image).
Figure 4.12: The welfare of matchings based on greedy approach for 100 riders with utilities from stationary distributions using truncated normal distributions (Legend same for all image).
Figure 4.13: Number of blocking pairs in matchings based on greedy approach learning algorithms for 50 riders with utilities from stationary distributions using truncated normal distributions.

Figure 4.14: Number of blocking pairs in matchings based on greedy approach for 100 riders with utilities from stationary distributions using truncated normal distributions.
Chapter 5

Conclusion

5.1 Conclusion

The utilities of the riders can be learned using reinforcement learning algorithms to provide a socially optimal matching among the riders. We present a reinforcement learning model that learns the expected utilities of the riders similar to a multi-agent multi-armed bandit. We compute the social welfare using utilitarian, egalitarian, Nash, and leximin ordering to provide the socially optimal matchings. The number of strong and weak blocking pairs is calculated for the socially optimal matchings. The $\epsilon$-greedy and UCB1 algorithms are implemented to learn the expected utilities. The $\epsilon$-greedy with $\epsilon = 0.1$ outperforms other algorithms in terms of accuracy in learning the expected utilities of the riders. UCB1 provides the socially optimal matching with higher social welfare for each social welfare concept. Socially optimal matchings computed using the $\epsilon$-greedy algorithm with $\epsilon = 0.1$ has a lower number of blocking pairs in general as compared to that of UCB1 algorithm suggesting more stability of the matchings. Since, we are trying to maximize the social welfare of socially optimal matchings, UCB1 performs the best since it provide maximum welfare for all social welfare concepts.
5.2 Future Work

The model implements $\epsilon$-greedy and UCB1 algorithm to learn the expected utilities of the riders. There are other reinforcement learning algorithms which theoretically, should provide higher accuracy for learning expected utilities [GM11, BGZ14]. The model could be extended to implement other reinforcement learning algorithms to analyze if they perform better than the algorithms implemented in the model with respect to learning the expected utilities, social welfare, and stability of the matchings.

The model provides the socially optimal matchings by computing the social welfare of all the possible matching using utilitarian, egalitarian, Nash, and leximin ordering. Calculating the Social Welfare for all possible matching in general is an NP-hard problem [RR10]. With the current computational power, the model can compute matchings for up to 14 riders. There are many ongoing research on the theoretical analysis and the approximation of social welfare [RE10, AKWX15, NRR12, BMV18]. One future direction is to extend the model to approximate the utilitarian, egalitarian, Nash, and leximin social welfare to find the socially optimal matching for a higher number of riders.
Bibliography


1962.


[RC19]  Saeid Rasulkhani and Joseph Y.J. Chow. Route-cost-assignment with joint user and operator behavior as a many-to-one stable


Appendices
Appendix A

Additional Results

A.1 Truncated normal distributions
Figure A.1: Percentage with error bars of 6 riders learning their best match with constant utilities using truncated normal distributions.
Figure A.2: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 6 riders with constant utilities using truncated normal distributions.
Figure A.3: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 6 riders with constant utilities using truncated normal distributions.
Figure A.4: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 6 riders with constant utilities using truncated normal distributions.
Figure A.5: Percentage with error bars of 6 riders learning their best match with utilities from stationary distributions using truncated normal distributions.
Figure A.6: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 6 riders with utilities from stationary distributions using truncated normal distributions.
Figure A.7: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 6 riders with utilities from stationary distributions using truncated normal distributions.
Figure A.8: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 6 riders with utilities from stationary distributions using truncated normal distributions.
Figure A.9: Percentage with error bars of 10 riders learning their best match with utilities from stationary distributions using truncated normal distributions.
Figure A.10: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 10 riders with constant utilities using truncated normal distributions.
Figure A.11: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 10 riders with constant utilities using truncated normal distributions.
Figure A.12: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 10 riders with constant utilities using truncated normal distributions.
Figure A.13: Percentage with error bars of 10 riders learning their best match with utilities from stationary distributions using truncated normal distributions.
Figure A.14: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 10 riders with utilities from stationary distributions using truncated normal distributions.
Figure A.15: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 10 riders with utilities from stationary distributions using truncated normal distributions.
Figure A.16: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 10 riders with utilities from stationary distributions using truncated normal distributions.
Figure A.17: Percentage with error bars of 14 riders learning their best match with constant utilities using truncated normal distributions.

A.2 Uniform distributions
Figure A.18: Percentage with error bars of 14 riders learning their best match with constant utilities using uniform distributions.
Figure A.19: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 14 riders with constant utilities using uniform distributions.
Figure A.20: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 14 riders with constant utilities using uniform distributions.
Figure A.21: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 14 riders with constant utilities using uniform distributions.
Figure A.22: Percentage with error bars of 14 riders learning their best match with utilities from stationary distributions using uniform distributions.
Figure A.23: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 14 riders with utilities from stationary distributions using uniform distributions.
Figure A.24: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 14 riders with utilities from stationary distributions using uniform distributions.
Figure A.25: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 14 riders with utilities from stationary distributions using uniform distributions.
A.3 Binomial distributions
Figure A.26: Percentage with error bars of 14 riders learning their best match with constant utilities using binomial distributions.
Figure A.27: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 14 riders with constant utilities using binomial distributions.
Figure A.28: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 14 riders with constant utilities using binomial distributions.
Figure A.29: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 14 riders with constant utilities using binomial distributions.
Figure A.30: Percentage with error bars of 14 riders learning their best match with utilities from stationary distributions using binomial distributions.
Figure A.31: The welfare of socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned for 14 riders with utilities from stationary distributions using binomial distributions.
Figure A.32: Blocking pairs in socially optimal matching computed using utilitarian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by $\epsilon = 0.1$ for 14 riders with utilities from stationary distributions using binomial distributions.
Figure A.33: Blocking pairs in socially optimal matching computed using utilitar-ian, egalitarian, Nash, and leximin ordering, respectively from expected utilities learned by UCB1 for 14 riders with utilities from stationary distributions using binomial distributions.
Appendix B

Code Snippets

```python
if np.random.rand() < epsilon:
    return np.random.randint(self.model.matchRiders)

_best_estimated_utility = np.max(estimated_utility)
return np.random.choice(np.where(estimated_utility == _best_estimated_utility)[0])
```

Figure B.1: $\epsilon$-greedy algorithm
if ride < self.model.matchRiders:
    _ucb_estimate = estimated_utility + np.sqrt((self.degree * np.log(ride + 1)) / (ride_count + 1e-5))
else:
    _ucb_estimate = estimated_utility + np.sqrt((self.degree * np.log(ride + 1)) / ride_count)
_best_ucb_estimate = np.max(_ucb_estimate)
return np.random.choice(np.where(_ucb_estimate == _best_ucb_estimate)[0])

Figure B.2: UCB1 algorithm

if distribution == DistributionType.TRUNCATED_GAUSSIAN:
    _low, _high = 0, 1
    _mu = np.random.uniform(0, 1)
    _sigma = np.random.uniform(0, 1)
    true_utilities = np.array(stats.truncnorm.rvs((_low - _mu) / _sigma, (_high - _mu) / _sigma, loc=-mu, scale=_sigma, size=self.matchRiders))
    for _ in range(1, self.num_rides):
        utility = np.array(stats.truncnorm.rvs((_low - _mu) / _sigma, (_high - _mu) / _sigma, loc=-mu, scale=_sigma, size=self.matchRiders))
        true_utilities = np.vstack([true_utilities, utility])
    return true_utilities

elif distribution == DistributionType.UNIFORM:
    true_utilities = np.random.uniform(0, 1, self.matchRiders)
    for _ in range(1, self.num_rides):
        utility = np.random.uniform(0, 1, self.matchRiders)
        true_utilities = np.vstack([true_utilities, utility])
    return true_utilities

elif distribution == DistributionType.BINOMIAL:
    true_utilities = np.random.binomial(1, 0.5, self.matchRiders)
    for _ in range(1, self.num_rides):
        utility = np.random.binomial(1, 0.5, self.matchRiders)
        true_utilities = np.vstack([true_utilities, utility])
    return true_utilities

Figure B.3: Sampling utilities from respective distributions

98
if len(_sride_list) < 2:
    yield []
    return
if len(_sride_list) % 2 == 1:
    for _sride_idx in range(len(_sride_list)):
        for result in possible_matching(_sride_list[:_sride_idx] + _sride_list[_sride_idx + 1:]):
            yield result
else:
    a = _sride_list[0]
    for _sride_idx in range(1, len(_sride_list)):
        pair = (a, _sride_list[_sride_idx])
        for rest in possible_matching(_sride_list[1:_sride_idx] + _sride_list[_sride_idx + 1:]):
            yield [pair] + rest

Figure B.4: Finding the set of possible matchings $M$
Figure B.5: Computing maximum welfare using utilitarian, egalitarian, Nash, and leximin social welfare
for _reader, _read_mate in matching_dict.items():
    for _r1 in self.utilities[_reader].items():
        if _read_mate != _r1[0] and self.utilities[_reader][_read_mate] < _r1[1] \
            and self.utilities[_r1[0]][matching_dict[_r1[0]]] < self.utilities[_r1[0]][_reader]:
            if [_reader, _r1[0]] not in strong_bp and [_r1[0], _reader] not in strong_bp:
                strong_bp.append([_reader, _r1[0]])
                blocking_pair1 += 1

        elif _read_mate != _r1[0] and (self.utilities[_reader][_read_mate] < _r1[1]
                and self.utilities[_r1[0]][matching_dict[_r1[0]]] <= self.utilities[_r1[0]][_reader]) or \
                (self.utilities[_reader][_read_mate] <= _r1[1] and
                 self.utilities[_r1[0]][matching_dict[_r1[0]]] < self.utilities[_r1[0]][_reader]):
            if [_reader, _r1[0]] not in weak_bp and [_r1[0], _reader] not in weak_bp:
                weak_bp.append([_reader, _r1[0]])
                blocking_pair2 += 1

Figure B.6: Computing number of blocking pairs in a matching