Mike Ehnot

Comparing One-Sided Matching Algorithms

**Introduction**

This project focused on simulating the performance of two different one-sided matching algorithms. A one-sided matching algorithm is an algorithm that matches a set of players with a set of items where each player gives a ranked preference list of all the items. These types of matching algorithms help allocate the items in a way that makes everyone happy.

The two matching algorithms used in this experiment are the serial dictatorship algorithm and the probabilistic serial mechanism algorithm. Serial dictatorship works by randomly selecting a participant and allowing them to choose their best available item. The rest of the participants are then selected randomly until all participants are matched with an item. The probabilistic serial mechanism is more complicated than the serial dictatorship. The probabilistic serial mechanism works by dividing each item into shares, and each player can select shares of their favorite available item until there are no shares left for any item. This part can be implemented using a simultaneous eating algorithm. Then, the number of shares each participant has for each item is used to calculate probabilities used to assign each item to a participant. So, if a participant has the most shares of an item, they are most likely to get the item (Hosseini).

These two algorithms were tested with two classes of participant preferences. One set of preferences was a uniform distribution of preferences where each participant’s preferences were a different random ordering of the items. The other class of preferences was a correlated preference list where every participant had a similar hierarchy of preferences in their list.

The goal is to measure the utility and variance for each combination of algorithm and preference class. The utility for a participant is defined as the index of the item that they were assigned in their preference list. For example, if a participant was given the first item in their list, their utility would be 1, and if they got the tenth item in their list, their utility would be 10. In this case a lower utility score is better. This experiment also measures the total utility of each simulation by summing the utility of all participants. The variance of each simulation’s utilities are measured as well. This is because there can be cases where simulations have similar total utilities, but different variances. For example, one simulation with a high variance may give half the participants their first choice and half their last choice, and another could give every participant their median choice. In that case the total utilities of both simulations would be the same, but the variance provides more context on the fairness of the simulation.

**Experiment**

The code for calculating a single simulation of this experiment begins by forming preference lists. The uniform preference list is made by assigning a random permutation of every item to each participant as their preference list. The correlated preference lists are made by assigning each participant a random permutation of all the items where certain ranges of the items are maintained. For example, a 40-item preference list would be a random permutation of 0 to 10 appended to a random permutation of 10 to 20 and so on. This ensures that every participant has some ordering of 0 to 10 as their top 10 choices and so on.

The next part executes the serial dictatorship algorithm. This is done by generating a random selection order by permuting the list of participants, and iterating through the selection order where each participant selects their best available item and removes it from the list of available items. This is done for both the uniform and correlated preference sets.

Then the probabilistic serial mechanism is performed. The first step is to create a list of available item shares. This experiment makes a list where there are 100 copies of each item to represent a share of each item. The shares are allocated in a similar way to the serial dictatorship where the algorithm cycles through each participant and they can select one share of their best available item. A matrix keeps track of the number of shares each participant has for each item. The algorithm keeps cycling through the participants until there are no more shares left. Next, the assignments are done by iterating over each item and using the set of shares for each item to determine the probability each participant has for getting the item. For example, if two participants both have 50 shares of an item, they both have a 50% chance of receiving the item. Once an item is assigned to a participant, that participant loses their shares in all other items. At this point, the probabilities have to be re-calculated because the total amount of shares does not sum to the same number anymore. Once the probabilities are re-calculated, the next item can be assigned based off the remaining players who have shares in the item. There is a case where none of the unassigned participants may have shares in an unassigned item. This happens when all the players who had shares in that item were already assigned another item. In this case, the algorithm just assigns that item to the first available participant.

Next, the utilities are calculated. This is done by looking at each assignment pairing and finding the index where that item is on the participant’s preference list.

Then, a bar graph is created to show the utility distributions for each algorithm/preference class combination. The x-axis is the participants sorted by their individual utility, and the y-axis is the individual utility of the participants.

Finally, the total utility and variance is calculated for each algorithm/preference combination. The total utility is calculated by summing the utilities of each participant, and the variance is the population variance of all the individual utilities.

This experiment is then simulated 100 times with new preferences and selection orders generated each time. A running total of each total utility and variance is kept so the average total utility and average variance can be calculated in the end.

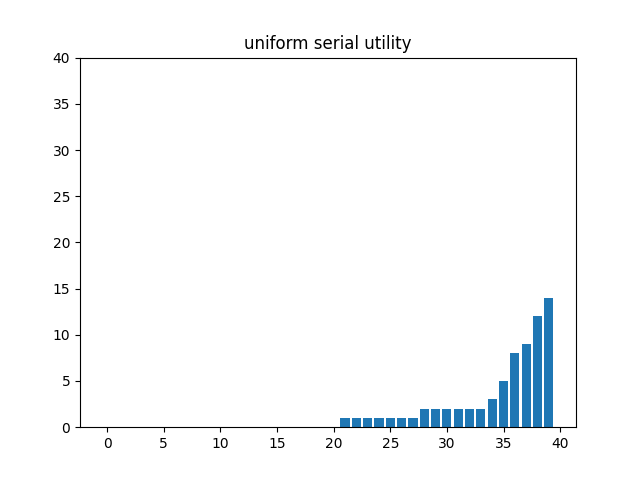
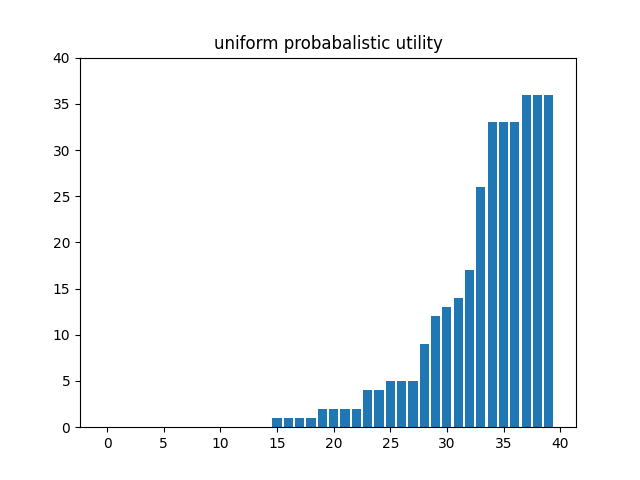
**Results**

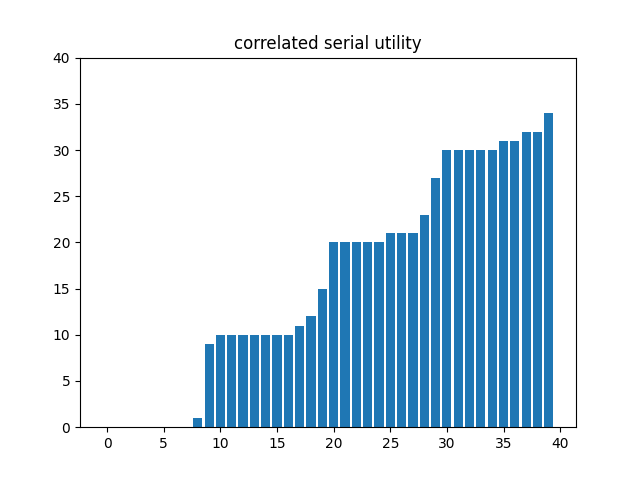
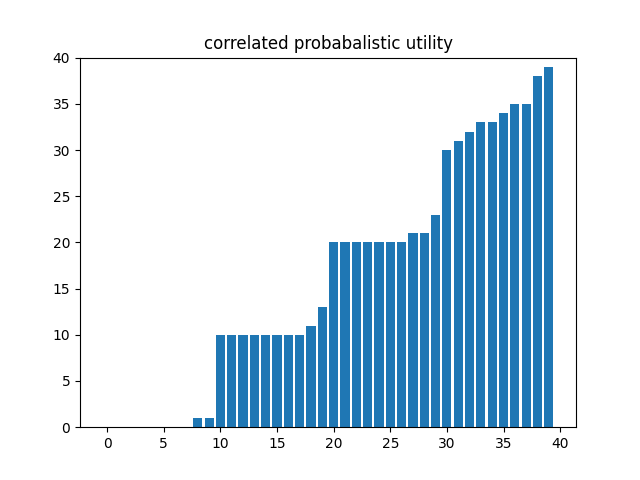
After 100 simulations the average total utilities and variances for each algorithm/preference combination were:

A black screen with white text

Description automatically generated with low confidence

The individual utility graphs were generated:





**Discussion**

As seen in the 100 simulations, the probabilistic serial mechanism performs much worse for the uniform set of preferences, but it performs roughly equal to the serial dictatorship for correlated preferences. Variance is always higher for the probabilistic serial mechanism, but it is significantly higher for uniform preferences.

The graphs help illustrate the differences in utility and variance between the algorithms and preference classes. Graphs with less blue show better utility and looking at the slope the graphs give an idea of how the variance changes. A higher slope means more variance, and flatter slope is less variance.

The most significant uncertainty of these simulations is the dilemma of properly using the probability matrix to assign pairs in a way where the player with the most shares in an object has the best chance of getting that object. This idea gets lost when there are items with no shares assigned to them and an arbitrary pairing is made. That case may be responsible for the probabilistic serial mechanism’s significantly worse performance for uniform preferences. The next section explores a better way of handling the probability matrix.

**Future Improvement**

After more research, I found that properly the probability matrix creates an intrinsic problem with the probabilistic serial mechanism. There are two ways of solving this problem. The first method is to not touch or re-normalize the probability matrix. This, however, allows for a participant to get multiple items or zero items, which is not suitable for this problem. The other way of solving this is to properly decompose the probability matrix into permutation matrices and probabilities. This can be done using The Birkhoff Algorithm. The Birkhoff algorithm accepts a “bistochastic matrix” (a matrix where each row and column consists of positive numbers that sum to 1) and returns a list of permutation matrices where each permutation matrix has a probability associated with it that represents the probability of that permutation matrix being the result. The permutation matrix represents the assignments of each item to a participant since each row and column only has one 1. Below is a small bistochastic matrix which is the same thing as the probability matrix.

A picture containing font, white, design

Description automatically generated([Wikipedia](https://en.wikipedia.org/wiki/Birkhoff_algorithm))

Birkhoff decompositions converts the bistochastic matrix into the form below.

A picture containing font, text, typography, handwriting

Description automatically generated([Wikipedia](https://en.wikipedia.org/wiki/Birkhoff_algorithm))

Now, probabilities can be used to determine which permutation matrix to use for pairings. For future work, it would be useful to see how using Birkhoff decomposition would affect the performance of the probabilistic serial mechanism.

**Bibliography**

Hosseini, Hadi, Kate Larson, and Robin Cohen. "Random serial dictatorship versus probabilistic

serial rule: A tale of two random mechanisms." *arXiv preprint arXiv:1503.01488* (2015).

Birkhoff algorithm. (n.d.). In *Birkhoff algorithm*.

<https://en.wikipedia.org/wiki/Birkhoff_algorithm>

Simultaneous eating algorithm. (n.d.). In Simultaneous eating algorithm. https://en.wikipedia.org/wiki/Simultaneous\_eating\_algorithm