Investigating the Performance, Efficiency, and Fairness in the Top Trading Cycles Algorithm for Generalized Agent-Object Allocation Problems

Elias Fizesan

Coding Markets [ECON BC2224] Spring 2023

Abstract

This paper investigates the performance, efficiency, and fairness of the Top Trading Cycles (TTC) algorithm in generalized agent-object allocation problems. We first provide a theoretical analysis of the TTC algorithm and develop a simulation framework to generate data for our empirical analysis. Our study focuses on factors such as algorithm convergence speed, agent utility, and the impact of preference submission order on allocation outcomes and fairness. Our findings indicate that the TTC algorithm exhibits logarithmic performance with respect to the number of agents, is Pareto efficient, and is strategy-proof, meaning agents cannot benefit from misrepresenting their preferences. Additionally, our analysis demonstrates that the TTC algorithm maintains a high degree of fairness in utility distribution, regardless of the order in which agents submit their preferences. These results support the widespread use of the TTC algorithm in various allocation problems, showcasing its effectiveness in providing fair and efficient allocations.

1.1 Introduction

The Top Trading Cycles (TTC) algorithm is a widely used method for allocating objects among agents in a fair and efficient manner. It has been applied to a variety of allocation problems, including school choice, kidney exchange, and resource allocation in cloud computing and networking. In this paper, we investigate the assignment factors that affect the TTC algorithm in generalized agent-object allocation by looking at factors such as algorithm convergence speed and agent utility. Our objective is to gain a better understanding of how the algorithm performs under different allocation scenarios and to show and explain several of the TTC's properties. Specifically, we aim to answer the following research questions:

1. How does the performance of the TTC algorithm vary across different market conditions?
2. How can the Pareto Efficiency and of the algorithm be explained through its algorithmic structure?
3. Does the TTC algorithm provide a strategy proofness for its participants?
4. How does the TTC algorithm perform under different fairness criteria, and what factors contribute to its fairness or unfairness?

To answer these questions, we first provide a theoretical analysis of the TTC algorithm and derive its properties for various allocation scenarios. We then develop a simulation framework to generate data for our empirical and statistical analysis.

2.1 The TTC Simulation

The Top Trading Cycles (TTC) algorithm is a mechanism for allocating indivisible goods to a set of agents who have preferences over those goods. The algorithm was first introduced by David Gale and Lloyd Shapley in 1962, and has since been used in a wide variety of allocation problems.

The basic idea behind the TTC algorithm is to allow agents to form cycles based on their preferences, and then to trade the goods within those cycles until no further trades can be made. Specifically, the algorithm proceeds as follows:

1. Each agent submits a list of their preferred goods, ranked in order from most to least preferred.
2. The algorithm identifies a set of cycles, where each cycle consists of a group of agents who each have their top-ranked good as their least-preferred good in the cycle.
3. Within each cycle, the agents trade their goods in a pairwise manner, with each agent giving their least-preferred good to the agent who values it most highly. This continues until no further trades can be made within the cycle.
4. Once all cycles have been identified and trades completed within them, any remaining agents who have not received their most-preferred good are added to a new cycle, and the process repeats.

The TTC algorithm has a number of desirable properties, including individual rationality (each agent receives a good that they prefer to their initial endowment), Pareto optimality (no further trades can be made to increase overall utility), and minimal envy (each agent values their own allocation at least as highly as any other agent's).

In our simulation, the core TTC algorithm code is split up into various parts, which have been outlined below depending on its functionality.

2.1.1 Iterate.m:

The iterate function in the simulation implements the Top Trading Cycles (TTC) algorithm to allocate objects to agents based on their preferences. The function takes two inputs: prefs, a matrix representing the preference orderings of agents for the objects, and allocations, a vector containing the initial allocations (usually initialized with all zeros).

The TTC algorithm proceeds as follows:

1. The function initializes a counter to keep track of the number of iterations.
2. It enters a while loop, which continues until all preferences are marked as -1, indicating that all agents have been allocated an object.
3. In each iteration of the loop, the function finds the most preferred object for each agent that has not yet been allocated. It stores these preferences in the most_prefs vector.
4. The detectCycles function is called to identify cycles in the directed graph formed by the agents' preferences. Each cycle represents a group of agents who can be allocated objects in a mutually beneficial way.
5. For each identified cycle:
    a. The cycle's nodes (agents) and edges (preferences) are extracted.
    b. For each agent in the cycle, their row in the prefs matrix is updated to -1, indicating that they have been allocated an object and are no longer participating in the algorithm. Additionally, any mention of the agent in the remaining preference lists is replaced with -1.
    c. For each edge in the cycle, the corresponding allocation is updated, assigning the agent their most preferred object.

The function continues iterating until all agents have been allocated an object, and all preferences are marked as -1. The final allocations and the number of iterations are returned by the function.

2.1.2 detectCycles.m:

The detectCycles function, which is called by the iterate function above, aims to identify cycles in a directed graph represented by the input array *arr*. A cycle is a sequence of nodes (agents) connected by directed edges (preferences) such that the first and last nodes in the sequence are the same.

1. The function initializes the length n of the input array, a visited boolean vector of length n to keep track of visited nodes, and an empty cell array cycles to store the detected cycles.
2. It iterates through the array, and for each unvisited node i with a preference not equal to -1:
    a. It initializes an empty cycle list and sets current to i.
    b. It enters a while loop, which continues until the current node is visited or its preference is -1.
    c. It marks the current node as visited and adds it to the cycle.
    d. It updates the current node to the next node in the preference list. If the current node exceeds the length n, the loop breaks.
    e. After the loop, the function checks if the cycle has been completed by comparing the first and last nodes in the cycle.

f. If a complete cycle is found, it extracts the unique_cycle by selecting the elements from the identified index to the end of the cycle.

g. The isContained function is called to ensure that the unique_cycle is not already in the cycles list. If it's not, the edgesFromCycle function is called to extract the edges from the unique cycle, and the resulting cycle and edges are added to the cycles list.

h. The isContained function checks whether a given cycle is already in the cycles list. It iterates through the list and compares the length and sorted elements of the input cycle with the stored cycles. If a match is found, it returns true; otherwise, it returns false.

3. The edgesFromCycle function extracts the edges from a given cycle. It iterates through the nodes in the cycle and, for each node, creates an edge between the node and its preference in the input array arr. These edges are stored in a cell array and returned.

The detectCycles function ultimately returns the detected cycles and their corresponding edges, which can be used to update the agents' preferences and allocations in the TTC algorithm.

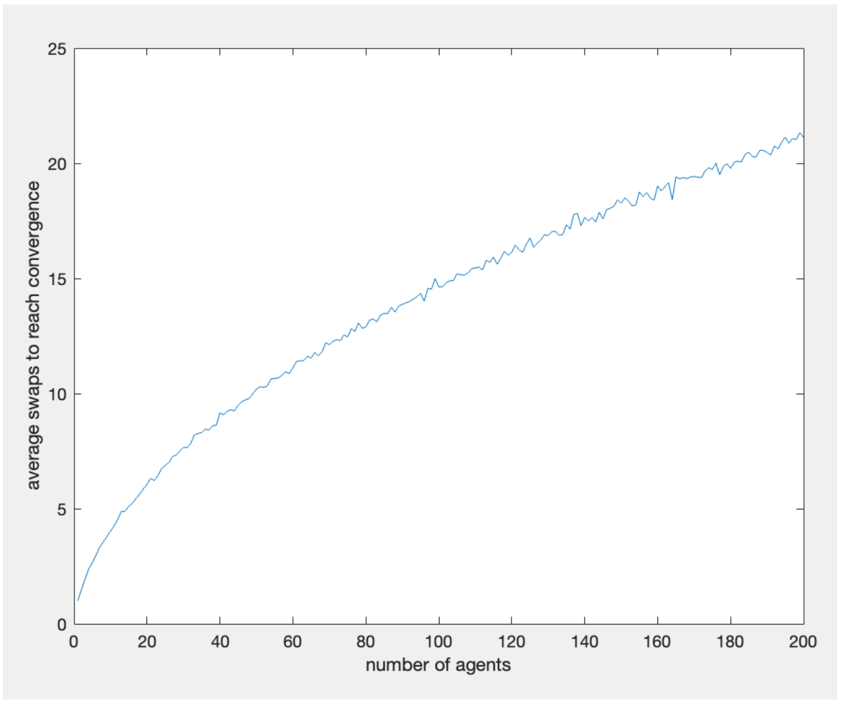
2.1.3 generatePreferences.m

The generatePreferences function takes two arguments: num_agents and num_objects. It returns a prefs matrix with dimensions num_agents by num_objects. This matrix represents the preferences of each agent over the available objects. The function uses a for loop to iterate over each agent in the prefs matrix. For each agent, it generates a random permutation of the available objects using the randperm function in MATLAB. The resulting permutation represents the agent's preferences over the objects, with the first element in the permutation being their most preferred object, and the last element being their least preferred object.

By generating random preferences in this way, the generatePreferences function simulates a scenario in which each agent has a unique set of preferences over the available objects. This is a common approach in allocation problems where agents have different preferences and no one agent's preferences are known in advance. The resulting prefs matrix can then be used as input for an allocation algorithm, such as the Top Trading Cycles (TTC) algorithm, to allocate the objects among the agents in a fair and efficient manner.

3.1 Analyzing Efficiency of TTC Algorithm

Now that we have a better understanding of how the code works, we can start to analyse the efficiency of the algorithm. Specifically, we will be looking at how fast the algorithm is able to converge to a stable state (a state where Pareto efficiency is achieved). To do this, we count the average number of times the detectCycles function is called and vary the number of agents and objects to measure their correlation. Plotting our data, we see the following results:



As expected, as we increase the number of agents, more iterations of the algorithm must be performed. However, one interesting thing to note here is the type or relationship between the variables – it seems to be logarithmic rather than linear. This makes sense because as the number of agents increases, the probability of having longer cycles that can be removed in one go increases. This reduces the number of swaps needed to reach convergence, which results in a logarithmic relationship with the number of agents.

4.1 Using Utility to measure aspects of the TTC algorithm

In the context of the TTC algorithm, utility can be defined as a measure of satisfaction or happiness that an agent derives from the allocation of an object. The preferences of each agent are represented as a ranking of objects, with higher-ranked objects providing higher utility to the agent. The utility that each agent receives can be quantified using different measures – in this simulations, we decide to use the ranking-based approach which is implemented in *calculateUtilities.m* as follows:

1. Assign a utility value to each object according to its rank in the agent's preference list. For example, if there are n objects, you can assign a utility value of n to the agent's most

preferred object, n-1 to the second most preferred object, and so on, until assigning a utility value of 1 to the least preferred object.

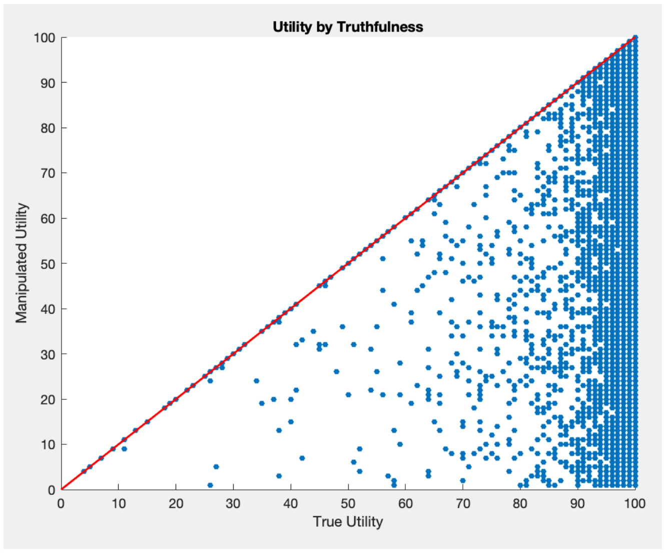
2. After running the TTC algorithm and obtaining the allocations, identify the object allocated to each agent.
3. Look up the utility value assigned to the allocated object for each agent according to their preference list.
4. The utility value represents the satisfaction or utility each agent derives from their allocated object.

4.2 Using Utility to Analyse Strategy Proofness

The strategy-proofness of the TTC algorithm implies that agents cannot manipulate their preferences to obtain a more favorable allocation. In other words, agents have no incentive to misrepresent their true preferences, as doing so would not result in a higher utility outcome. This property is essential for ensuring that the allocation process is fair and efficient, as it prevents agents from strategically altering their preferences to gain an unfair advantage.

To see whether the TTC algorithm is indeed strategy proof, we analyze the allocation process of the TTC algorithm. A new function, *manipulatePreferences.m*, is created which aims to generate 'fake' preferences that a randomly assigned agent will use. The algorithm is run once with the actual preferences, and another time with the 'fake' preferences. In each case, the utility is measured for that individual to see which preferences yield a higher utility.

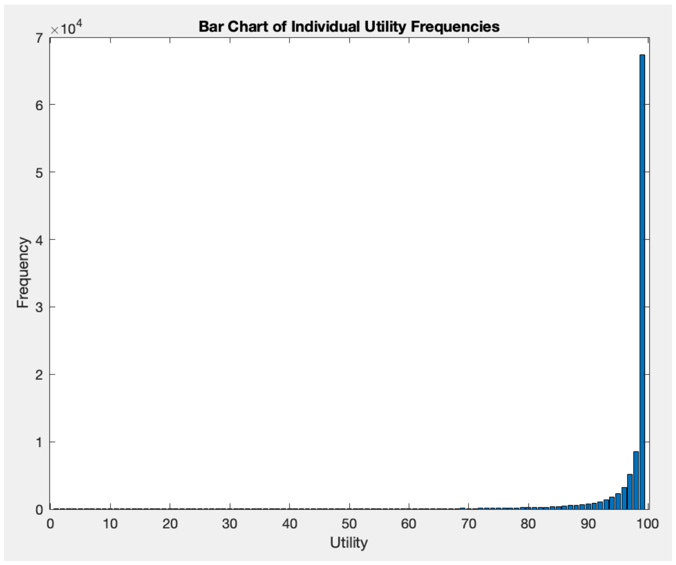
To anaylse the data, a scatter plot, where each dot represents the utility received from manipulated and true preferences, is sketched after 1,000 iterations of the algorithm where each simulation contains 100 agents (99 truthful and 1untruthful). In addition to this, a y=x plot is drawn. If the TTC algorithm is truthful (the agent will never be able to get a higher utility by manipulating their preferences), we should see that no dot lies above the straight line. The figure below empirically shows that the TTC algorithm must be strategy proof as there is no benefit of lying about preferences.



4.3 Anaylsing the goal of maximising each individual's utility

Pareto efficiency is an important concept in welfare economics, as it represents a situation where resources are allocated in a way that maximizes overall welfare without disadvantaging any individual. Empirically, Pareto efficiency of the TTC algorithm can be tested by examining the utility of individual agents for a given allocation. If an allocation is Pareto efficient, then no agent can increase their utility without reducing the utility of another agent. This means that the total utility of all agents in the allocation is maximized, and there is no alternative allocation that would result in a higher total utility.

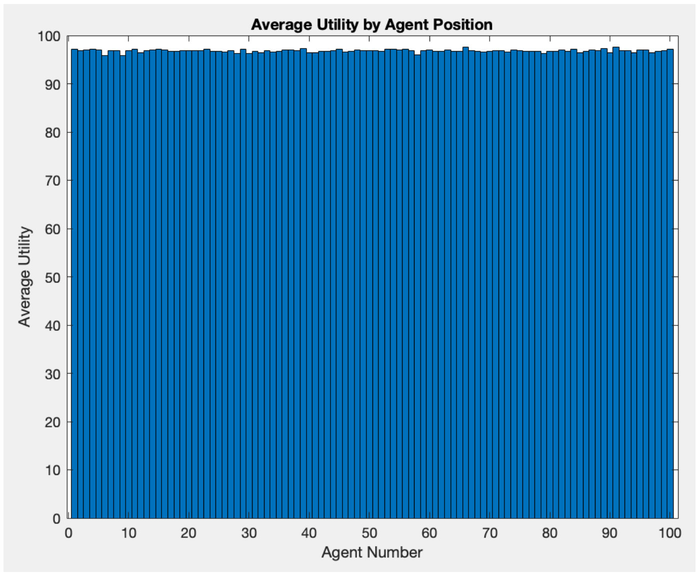
To create a simple visualisation to see whether utility is indeed maximised, we can statistically analyse the distribution of agent utilities. The barchart below shows the plotted data from running a simulation that contains 100 agents, 1000 times (notice that the frequency is in x10^4). We can see that it is heavily skewed to the right, meaning that most agents had a near 100 utility, with the chart dropping exponentially as we look at the frequency of agents who had a lower utility. To see the significance numerically, the data contained the following statistics: std of 8.894, interquartile range of 2, and a median of 100. This data clearly provides evidence to that fact that the TTC algorithm is Pareto efficient.



4.4 Measuring TTC Fairness Using Utility Analysis

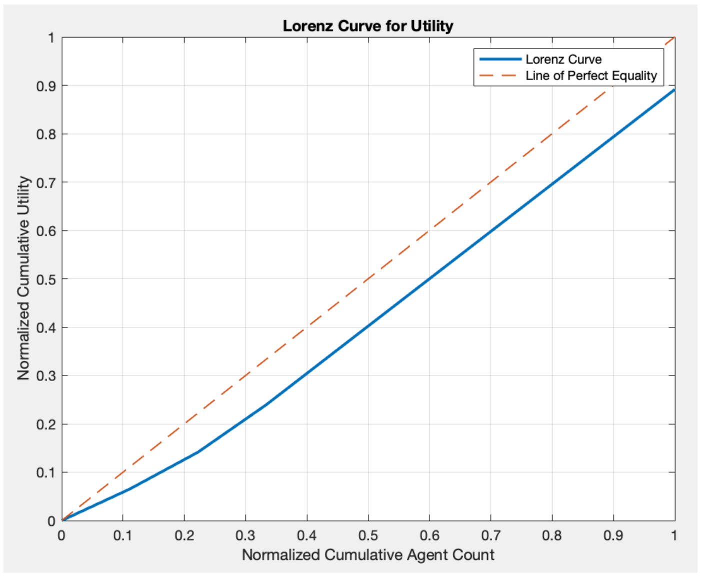
Our investigation of the fairness of the Top Trading Cycles (TTC) algorithm focuses on the impact of the order in which agents submit their preferences on the distribution of utility. To assess this aspect of fairness, we conducted a simulation to analyze the utility distributions across various allocation scenarios, taking into account the position of the agents when submitting their preferences.

The simulation results revealed a uniform distribution of utility, indicating that the TTC algorithm is indeed fair in this regard. This uniformity suggests that the order in which agents submit their preferences does not significantly influence the allocation outcomes, and consequently, the utility derived by the agents. This finding further reinforces the TTC algorithm's robustness in ensuring fair allocations, as it demonstrates its ability to maintain equitable utility distributions regardless of the order of preference submissions.



In our study of the fairness of the Top Trading Cycles (TTC) algorithm, we further examine the allocation outcomes using the Lorenz curve as an analytical tool. The Lorenz curve is a graphical representation of the distribution of a variable, typically used to assess income or wealth inequality. In the context of the TTC algorithm, we utilize the Lorenz curve to visualize the distribution of utility across the agents in various allocation scenarios.

Our simulation results revealed that the Lorenz curve (shown below) is nearly perfectly linear, which indicates a high degree of fairness in the TTC algorithm. A linear Lorenz curve signifies that the distribution of utility is highly equitable, with each agent receiving a proportionally similar share of utility. This observation supports the conclusion that the TTC algorithm maintains fairness in the allocation process, as it ensures an even distribution of utility among the agents, regardless of their preferences or other factors.



5.1 Conclusion

In conclusion, this paper provides a comprehensive investigation of the Top Trading Cycles (TTC) algorithm in the context of generalized agent-object allocation problems. Our theoretical analysis and simulation framework allowed us to examine the performance, efficiency, and fairness of the TTC algorithm under various market conditions and allocation scenarios. We assessed the algorithm's convergence speed, Pareto efficiency, strategy-proofness, and fairness using utility analysis and the Lorenz curve.

Our findings demonstrate the robustness of the TTC algorithm in delivering fair and efficient allocations. We observed a logarithmic relationship between the number of agents and the number of iterations required for convergence, indicating that the algorithm performs efficiently under different market conditions. The TTC algorithm was shown to be strategy-proof, as agents cannot benefit from misrepresenting their preferences, ensuring fairness in the allocation process.

Furthermore, our analysis revealed that the TTC algorithm is Pareto efficient, maximizing overall welfare without disadvantaging any individual. We also demonstrated the fairness of the TTC algorithm through the uniform distribution of utility and the nearly linear Lorenz curve, indicating an equitable distribution of utility among agents regardless of their preferences or the order in which they submit their preferences.

Taken together, these results highlight the effectiveness of the TTC algorithm in addressing agent-object allocation problems in various contexts, reinforcing its applicability in real-world scenarios where fairness and efficiency are crucial concerns.

6.1 References

https://economics.yale.edu/sites/default/files/leshnolo_-_simple_ttc.pdf

https://people.cs.umass.edu/~sheldon/teaching/mhc/cs312/2014sp/Slides/top-trading-cycles.pdf

https://www.cis.upenn.edu/~aaroth/courses/slides/agt17/lect11.pdf