Daniel McCown & Kyle Cooper

Repository: cooper

CSE 544T: Course Project

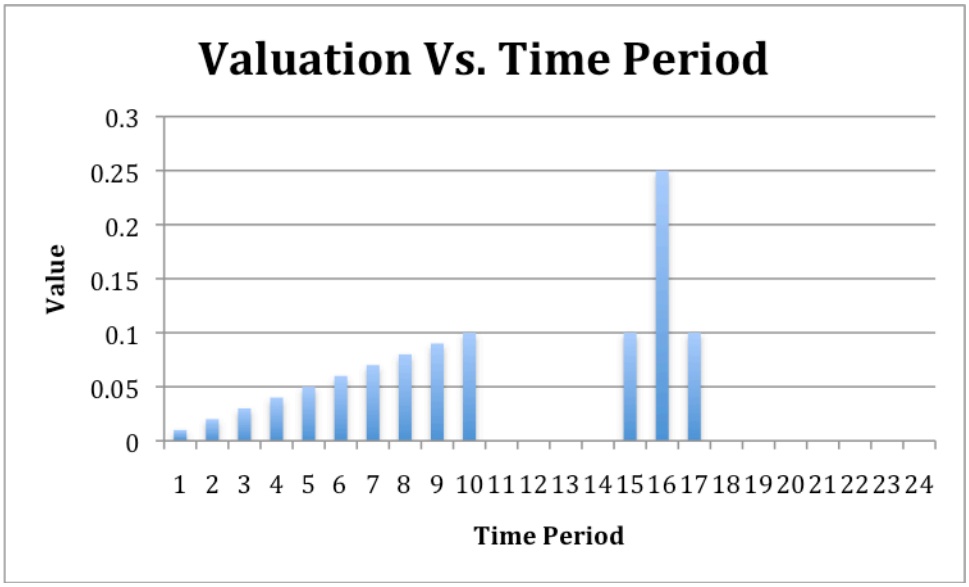
Problem

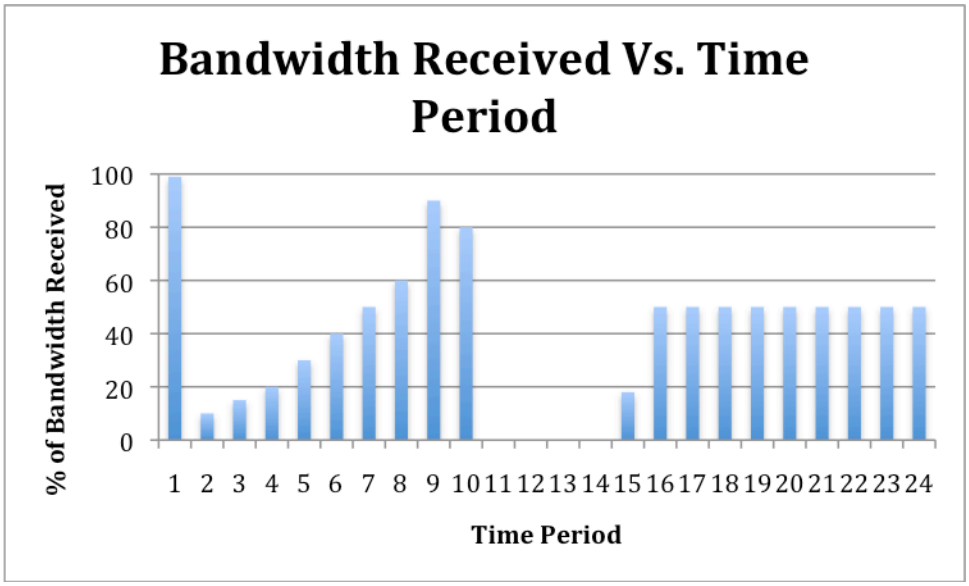
High Level

We wanted to approach the problem of how to fairly give people internet bandwidth in the times they desire it most. More specifically given peoples preferences over a 24 hr interval (valued per hour) we attempted to find a horizontal "slice" of the bandwidth that reflected a "fair" (proportional) division.

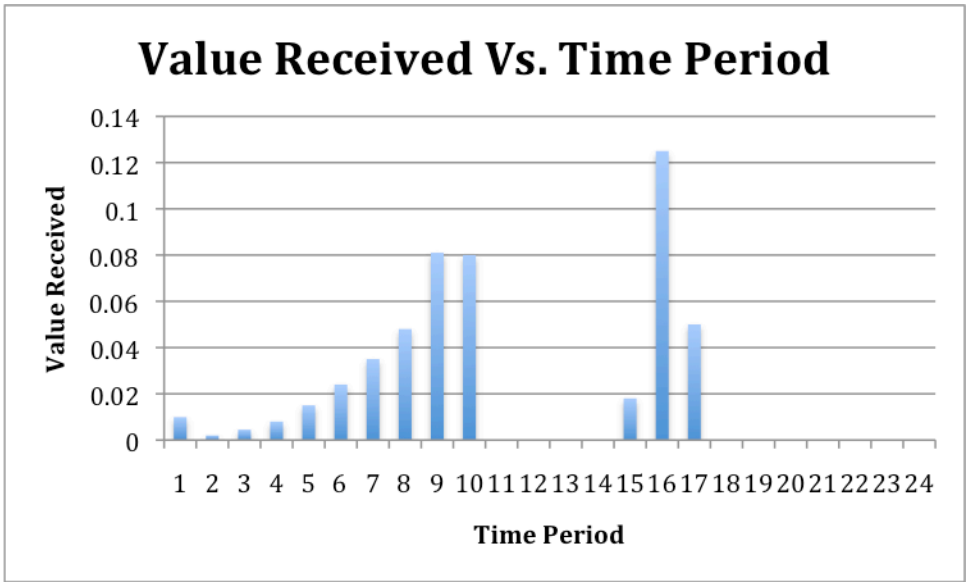
More Information

Working from the high level problem above we give the following graphs for a user. Shown below are first the user's value chart and then the user's fair slice given 2 players:





These two charts however do not show value received by the person from the bandwidth. This comes from multiplying their percentage received by their valuation of the time period to yield a perceived valuation. This graph is shown below:



These graphs along with the opening explanation should give a strong idea of the problem. We will further develop the nuances of how to achieve fair valuation and analyze the results of our algorithm.

Implementation:

The algorithm can be logically defined into two main parts. There is a preprocessing stage and division stage.

1. The pre-processing stage is simply transforming the inputted user preferences to an array where we see the value per "time chunk". For example given the person mentioned in the problem presentation, this preprocessing then gives an array that holds a different value for each of the 24 time chunks as seen in the Valuation vs. Time Period chart above.

2. The division stage has many sub-processes but at its core it is an implementation of Fink's last diminsher.

   In the following steps a "slice" consists of an amount of bandwidth during each time chunk. For our implementation we use 24 chunks to represent each hour in a day.

   We broke the problem into the following parts:

   a. Evaluating a slice:

      i. We simply sum up the (bandwidth at timechunk)*(value of timechunk) over all time chunks in a slice.

   b. Per Fink if the slice is valued higher than 1/n we cut the slice to 1/n:

      i. To cut the slice we first randomly choose a column then we attempt to cut. We then a remove a part of the bandwidth equal to (1/n * value of time chunk) thus we should be able to take from nearly all columns and are unlikely to have a column that has 0.

      ii. The decision to try to leave no column with 0 is not a decision based on fair division where utility is all that matters but rather an observation on real-world internet usage where one would like at least minimal internet access at all times in case of unplanned activities.

      iii. If there is less than (1/n*value of timechunk) value exists we simply take all of the value that is available.

      iv. This process continues until the user values the piece at exactly 1/n

         1. Note 1: Since this is an exact algorithm our "resolution" is the computers floating point resolution.

2. Note 2: that a running difference between initial valuation and the amount cut away makes it so that we do not revalue the slice between each while loop iteration.

c. Finally the slice is assigned and the process starts again with the player removed. When all but one player has received bandwidth then the final person gets whatever remains.

3. As an added feature we do post division envy analysis but this is not necessary to the algorithm.

Analysis:

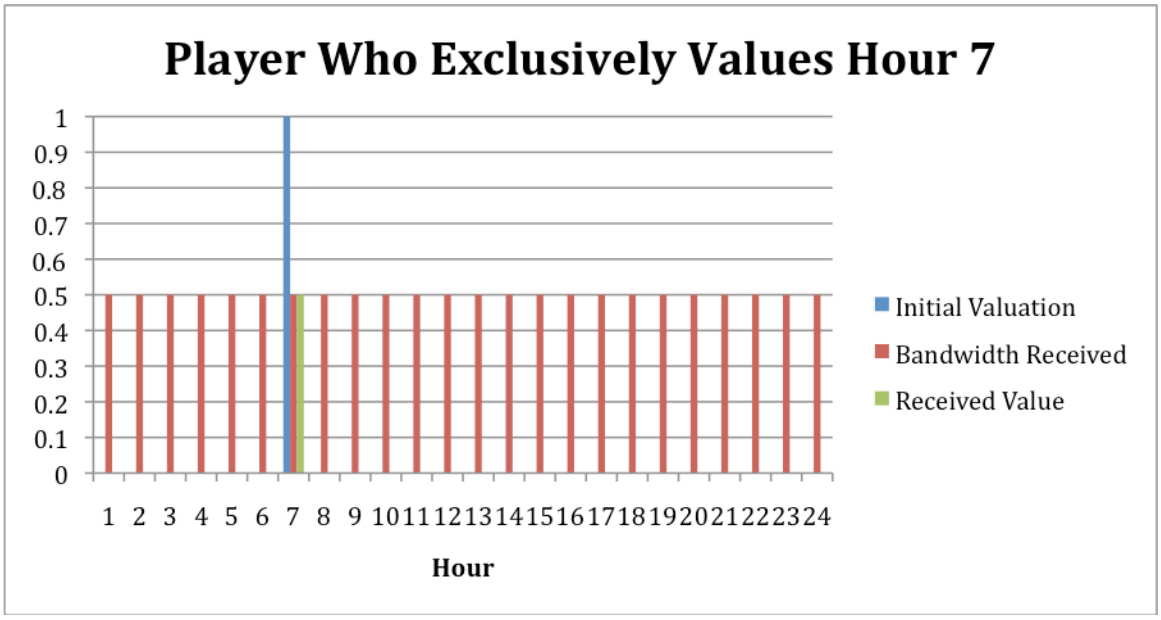
Theoretical

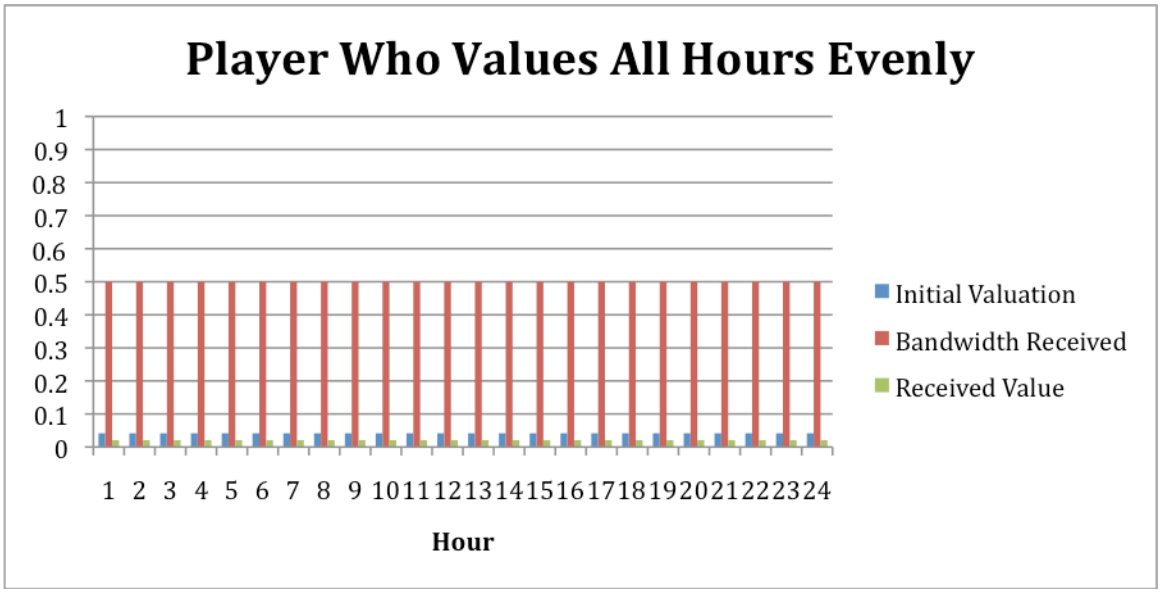
There were two main points that came up in the theoretical analysis of this algorithm.

1. That our implementation of the algorithm was $O(n^4)$. This is due to the fact this algorithm is originally $O(n^3)$ but our cut process can take up to $O(n)$ time.

2. Upon further examination we noted that cake and internet bandwidth are very different goods. Namely that for a single point in time all units of internet bandwidth have the same value to a player while for a single crumb wide slice of the cake not all crumbs may have the same value to a player.

   Given the above a slice of the bandwidth that is 1/n at all times would yield a proportional every player. It should be noted that this algorithm is envy-free and equitable but the sum of values for all players will never be greater than 1. We chose not to implement the algorithm as such because we wanted to evaluate an algorithm with wider applications. Furthermore our algorithm does generally create a higher total evaluation across all players because the last player generally gets a larger than 1/n slice. See the graphs below for an example given two players.





Tools Used for Runtime Analysis

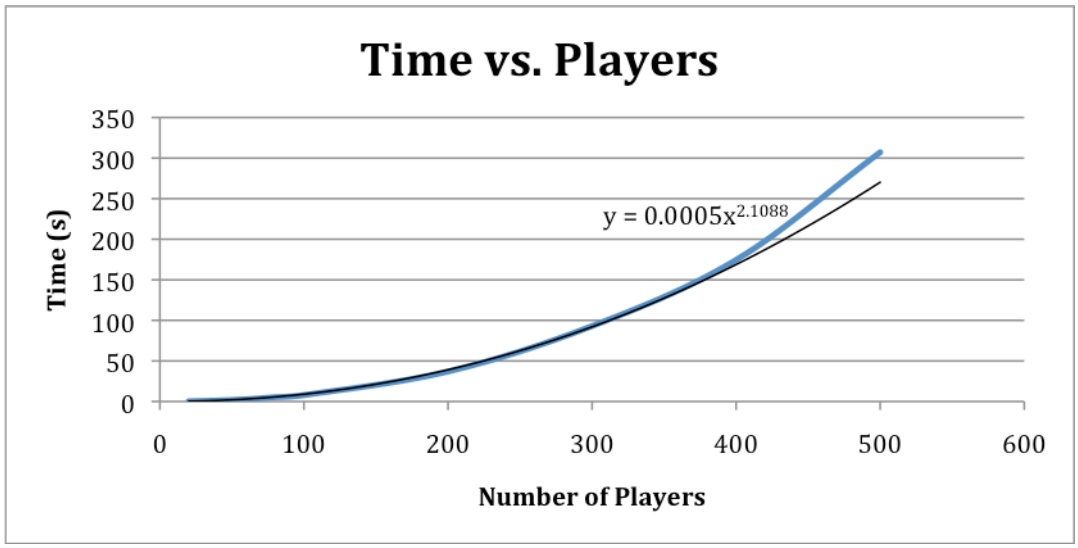
To analyze the run-time efficiency of our algorithm we generated random player sets of up too 500 players. We would then give these to our algorithm and use the cProfile package to determine the running time and number of function calls.

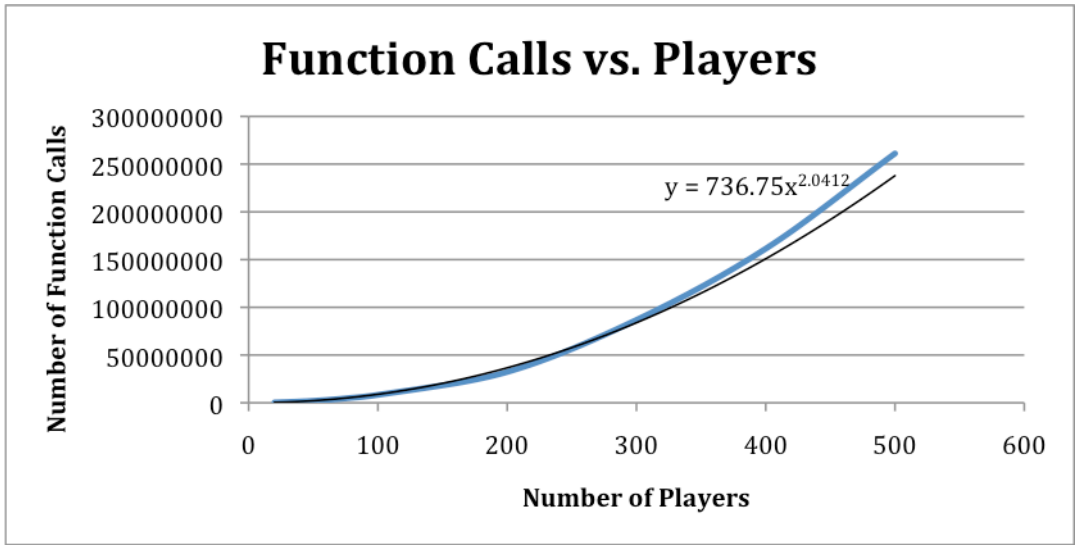
To further understand these numbers we graphed them using Excel and did a least squares regression to determine the exponent which it was following in runtime.

Performance Results

Using the previously mentioned analysis tools we gathered data to analyze the real world running time of our algorithm. Despite having a theoretical running time of

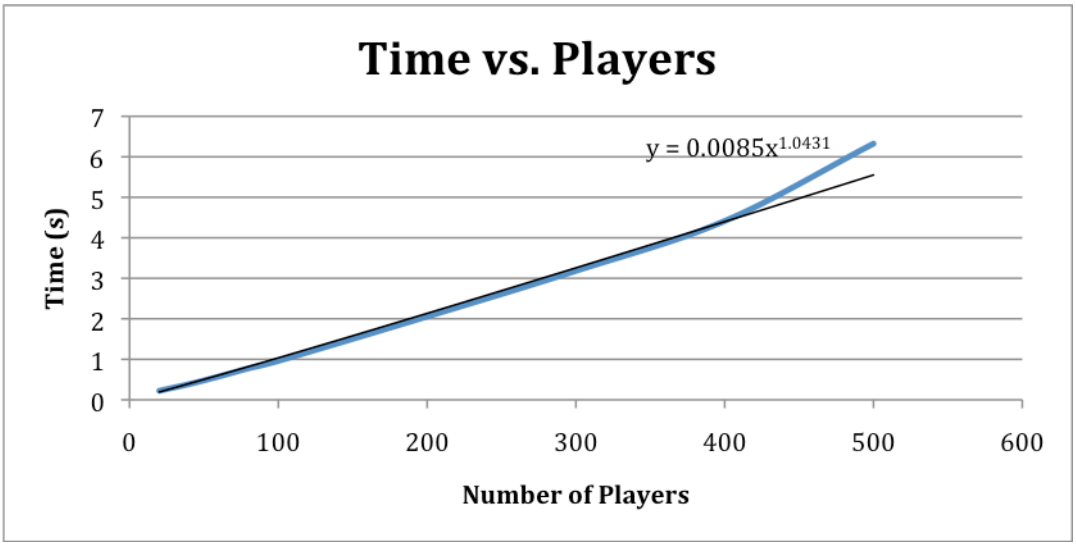
$O(n^4)$ our results showed an actual running time of just over n^2. This is because the worst case is each player cutting every allocation of internet bandwidth that is passed to him or her and given the structure of our cut algorithm in the worst case it also adds a factor of n to the running time. However, in general only a few players each round will cut the bandwidth and the cutting generally finishes quickly as well. To get the worst case would require very specific preferences and for the players to be in a specific order as well.

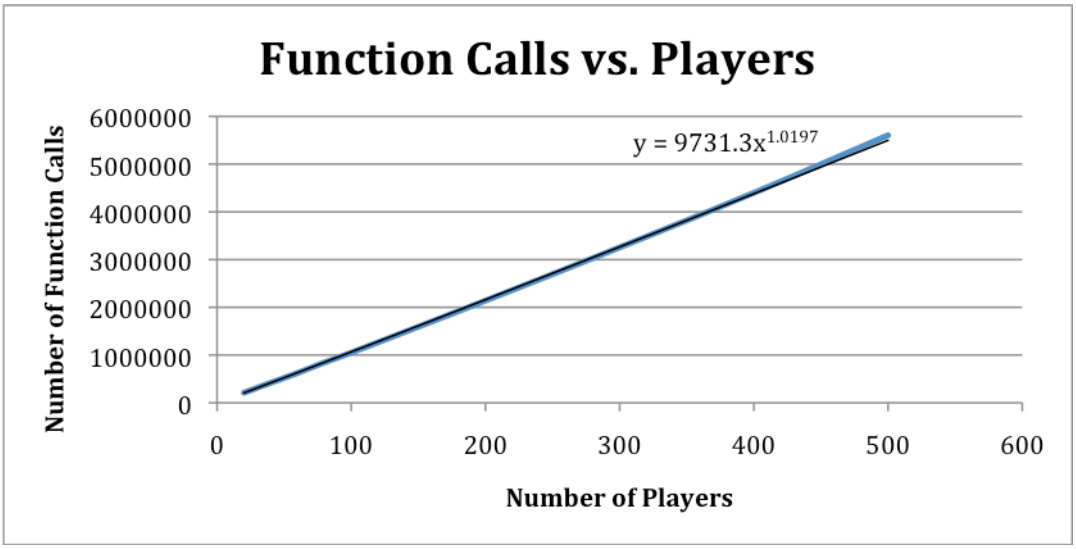




Performance Results: Alternate Cutting Algorithm

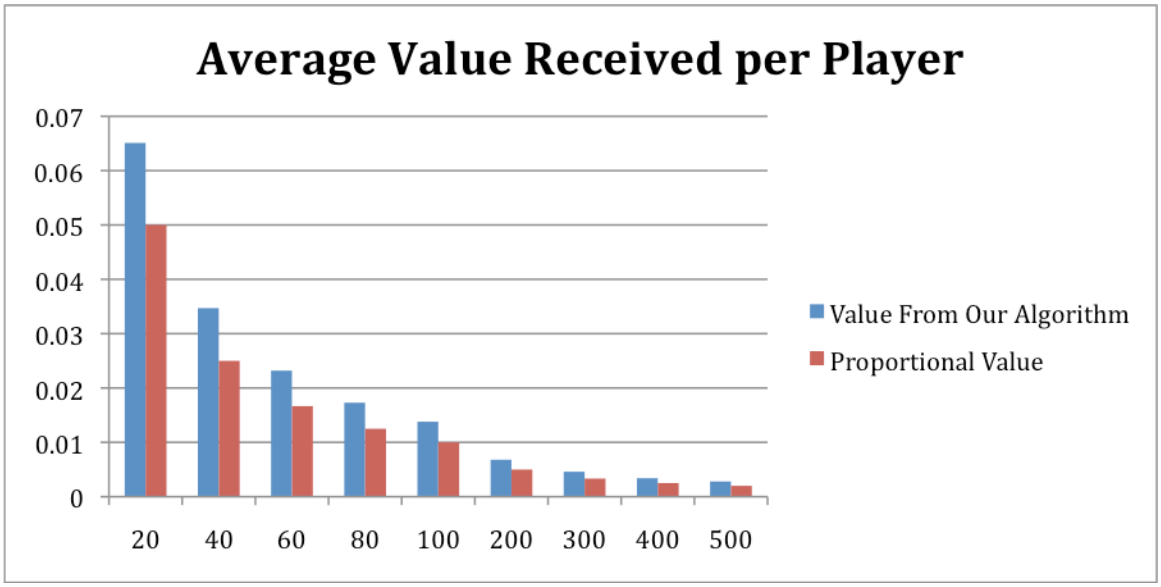
We developed an alternate cutting algorithm which did not attempt to ensure that all players received at least some bandwidth for every time chunk. Therefore it would cut at most the number of timechunks, in our case 24. This brings the total asyptomtic running time of the algorithm back to $O(n^3)$. The only downside is that players may have no bandwidth for many of the time chunks. Below are the runtime analysis results we gathered using this approach.





Fair Division Results

We verified our results manually for many small samples and then took the average value received per player for larger results. Every player except the last one to receive bandwidth, receives exactly 1/n in their own eyes. The last player receives at a minimum 1/n and in practice generally much more. The last player in fact receives so much more that this one player's received value significantly raises the average value received per player as can be seen in the graph below.



As mentioned above in reality only one player receives more than the proportional value. For a future implementation we would like to try an approximately envy-free algorithm where players may reenter the fray. Our current algorithm is not envy free and is also not equitable.

Conclusions

In conclusion we see that our algorithm generates at a minimum a proportional share of bandwidth. However, generally there will be one player that receives much more than the proportional value, so much so that the average is largely offset by that one player. Our algorithm runs generally much faster than the worst-case $O(n^4)$ running time, but still is slow as n grows larger than 100.

Lastly, there exists a naive algorithm that is asymptotically faster, proportional, and envy-free but does not add any additional value over the proportional value. This algorithm takes advantage of the difference between bandwidth and cake.