Abstract

The Moving Knives algorithm attributed to Austin provides a method to divide cake between two parties in such a way that each party feels they have received exactly half the value of the overall of the cake. The solution provided by the algorithm is proportional, envy-free, and equitable. The algorithm is written for preferences that are both independent and additive. Since many scenarios in life involve preferences which do not have these characteristics and have dependencies within the preferences, this project is focused on modifying the Moving Knives algorithm to handle these cases. After experimentation with preprocessing the value function input and modifying the way the algorithm evaluates potential divisions, a variation was produced that searches for a division location that yields proportional results while ensuring necessary dependencies are met. The resulting algorithm however does not guarantee such an outcome is available. It only applies a single method in an attempt to find one.
**Austin’s Moving Knives**

During the semester, we discussed the Moving Knives algorithm attributed to Austin for dividing cake for distribution to two receiving parties. The objective of the algorithm is to find division locations in the cake such that the resulting pieces will be viewed as having equal value to the recipients. After cutting the cake at these locations, the pieces of cake can be assigned while guaranteeing a proportional, envy-free, and equitable outcome. That is each party believes that they received at least half of the cake’s value (proportional), each party views their received portion as no less valuable than the other party’s (envy-free), and the perceived value by each party of their own portion is the same (equitable).

The algorithm is carried out by first moving a single knife over the cake, from one end to the other. While the knife is moving, one of the two parties (or a mediator with knowledge of their preferences) says “STOP” as the area the knife has previously passed over reaches a value of 50% of the total cake. Then another knife is added at the first’s starting position and the knives are moved together such that the section between the knives is maintained at 50%-value, in the eyes of the party who said “STOP”. The party who has yet to say “STOP” is then instructed to do so when he/she also perceives the value of the area between the knives as 50%. The knives are held at their current positions and the cake is cut, yielding the section between the knives for one recipient and the remaining section(s) for the other. The allocation of which player gets which at this point is arbitrary since each player sees each of the two available portions as 50%-value.

The algorithm works by exploiting the continuous nature of the parties’ preferences, or value functions, for the cake. The Intermediate Value Theorem provides a guarantee that while the pair of knives is moving together, the second party will see the interior section’s value reach 50% at some point. Further, the algorithm depends on the independent and additive nature of the preference values assigned to the various points along the cake. As the interior section changes to include a specific region, the new...
region’s value is added into the total interior value regardless of which region is being left behind with its value subtracted from the total interior value.

As discussed in class, there are numerous real-world applications for this algorithm. One that I found particularly interesting was the division of time. For example, consider a scenario in which an individual is returning to his hometown for a short visit. Let’s suppose that during the visit, he will likely want to spend social time with two separate groups of people, friends and family members. For the sake of argument, we’ll treat each group as a unified whole party. Due to these groups’ schedules and the preferred activities, say dinner with family and late drinks with friends, the two groups would have different values on different portions of the traveler’s time. So the Moving Knives algorithm could be applied to come up with a time division in which each of the two groups could be allocated a perceived 50%-value segment of the traveler’s schedule.

Just like when applying the algorithm to cake, the assumptions must be made that the timeslots are valued independently and the function representing the value as a function of the timeslots is continuous. When these are true (perhaps this would require approaching infinitely small timeslots), the algorithm can be applied without significant modification – traveler requests group preferences and carries out the process as the mediator.

However, there are many circumstances in which these assumptions are not true. Frequently, when looking at time management and division of resources, there are interrelationships between the timeslots, meaning the preferences are no longer additive or independent. Consider for example a set of five tasks that each requires four timeslots worth of attention. The stakeholders of those tasks would likely argue that if they were only allocated the resources to work during 10 timeslots leading up to a deadline, their perceived value would be less than 50%, because while two of the tasks could be completed, a half-finished third task may be of no value at all.

These scenarios with dependencies impacting the preferences and perceived values are not conveniently handled by any of the algorithms that we discussed in class. Therefore, in this project, I have attempted to explore a possible modification of Austin’s Moving Knives algorithm to find a proportional, envy-free, and equitable solution to these types of scenarios.

Establishing a Baseline

In order to create an environment in which I could simulate and test a modification to Austin’s algorithm in software, I first chose to implement the algorithm directly as a baseline.

In an effort to follow the algorithm as closely as possible, I chose to simulate the movement of the knives across the cake by advancing their position step-by-step based on a resolution parameter. At each location, I evaluate the necessary players’ perceived values of the interior section to decide whether or not to have the player say “STOP,” -
both players when one knife is moving, and the “quiet” player only once both knives are moving. This implementation yields results that are nominally 50% for each player (smaller resolution values yield closer to 50-50% outcomes, but increase execution time).

Accepting that the results are within tolerance of the desired outcome, I felt that my direct Moving Knives implementation sufficiently represents the behavior of the algorithm and deciding to begin experimenting with modifications.

Preprocessing Preferences

While thinking about how I could modify the preference file structure to denote dependencies, I began to consider the possibility of just adding a preprocessing step to the algorithm. This preprocessing step would take in the preference and dependency data and create a new value function that encapsulated the dependencies. Then this new preference data would be run through the baseline Moving Knife algorithm.

The two methods I experimented with as I explored this approach both involved a sort of “impact assessment” for a given point. For the first method, I simply assumed that all preference points were inherently dependent on those on their left. Like a series of falling dominoes, if one interval was not allocated, then all intervals after it would be of no value. The preprocessing I applied to represent this was to increase each preference value by the sum of all preferences to its right. This action effectively just created a value function with a downward slope for both players, leading to results that were rather unsatisfying. The simulations had each player biased to saying “STOP” extremely early, focusing on getting the first available intervals. I assessed that this didn’t really address the scenario as I was hoping since the increased desire of the early intervals significantly reduced the player’s relative preference for later intervals. While at a glance this could be misinterpreted as “working ahead” in the five task scenario described above, the early timeslots are equally as productive as the later ones, so their relative importance must be kept within reason to avoid abandoning the later timeslots altogether and still failing to complete the necessary tasks.

The second preprocessing approach didn’t yield much better results, as I tried to keep the relative preferences for the early intervals from growing too much. I decided to indicate specific dependencies (e.g. interval 0.75-0.80 depends on interval 0.25-0.30) before preprocessing and then just adding the preference for the dependent interval to that of the depended-upon interval. Again, this created too much of a skew in the relative value of the early intervals compared to the later ones.

Considering Dependencies before Saying “STOP”

After working with the preprocessing techniques to no avail, I decided it was time to work the dependencies into the algorithm itself. The idea I chose to implement was that for a given interval, the perceived value of that interval is only appreciated by the recipient if all dependencies that interval had are also met. For illustration purposes, consider the three intervals 0.10-0.15, 0.15-0.20, and 0.20-0.25. Also, assume that there
are dependencies such that the interval from 0.10 to 0.15 has a dependency on 0.00 to 0.05, and the interval from 0.20 to 0.25 has a dependency on 0.15 to 0.20. Finally, consider the question to a player regarding his perceived value of the segment from 0.10 to 0.25. In this scenario, he would exclude his value for 0.10 to 0.15 from his total because the other interval it depends on is not within the segment in question. On the other hand, the dependency for 0.20 to 0.25 is met within the segment, so its full value would be considered. Intervals with no dependencies (e.g. 0.15 to 0.20) would always be represented with their full value.

When I implemented this in the software, it became clear that there would be significant differences between this algorithm’s behavior and the baseline Moving Knives algorithm. The obvious initial change was checking for dependencies and not calculating the value gained for an interval whose dependencies were not met. But beyond that, I could no longer have the second player say “STOP” when he too saw 50% value between the knives. Because now that there are dependencies, there is no guarantee that 50% value lies outside the knives at that same instant in time. This meant I needed to have the second player watch outside the knives once the second knife was added.

Also, this change in the value calculation for a specified segment made the function representing the value as a function of knife positions no longer continuous. This occurs because as an interval that was being depended upon is left behind by the knife, the interval depending upon it suddenly drops to a value of zero. The possibility of these sharp drops disrupts the Intermediate Value Theorem and therefore removes the guarantee of an available proportional division. Regardless, I decided to continue down this path and see where this concept would take me.

**Testing**

Modifying the random preference generator, I was able to randomly incorporate dependencies as well. I used this to run a series of 5000 tests in which approximately 10% of the intervals in the value function had a dependency on at least one other interval. In these tests, I was able to see that 67% of the time, a proportional outcome was found
using this new algorithm. In each of these cases, the result was envy-free, as implied by proportionality for two parties, but not reliably equitable. The lack of equitability stems from the lack of continuity in the value function as the knives move since the first player’s value won’t stay right at 50% and the second player’s will likely “jump” over it. The remaining 33% of the time, the algorithm did not identify a possible proportional outcome. It is important to note that this does not mean that a proportional outcome is nonexistent for those test cases, it simply could not be found by this particular algorithm.

<table>
<thead>
<tr>
<th>Tests Run</th>
<th>Found Solution</th>
<th>Did Not Find Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>3327</td>
<td>1673</td>
</tr>
<tr>
<td>100%</td>
<td>67%</td>
<td>33%</td>
</tr>
</tbody>
</table>

Testing Results

Conclusion

My experimentation with the Moving Knives algorithm, from implementing it directly in software to modifying it to deal with dependencies in preference data, led me to a deeper understanding of what makes the algorithm work. It became clearer to me why the assumptions of independent and additive values are so significant, because without them, there’s no guarantee of a continuous value function and the benefits that brings. Since my modification removes the guarantee that a proportional outcome exists in this treatment of dependencies, it should be viewed as just a tool that may or may not provide a solution.