CSCI5454 - Design and Analysis of Algorithms
Crowd-Sourced Lecture (CSL): Fair Division

Mario Barrenechea
mario.barrenechea@colorado.edu

Daniel Peterson
daniel.peterson@colorado.edu

April 11, 2012

1 Introduction to Fair Division

Fair division is a difficult and practical problem that arises regularly in our everyday lives. The premise is that there are several people who want to divide something (a good or resource) up so that everyone gets their *fair share*. A fair share is determined by the person - they should think they received as much as anyone else. The thing to split may be something that everyone wants (e.g. money or food) or that everyone wishes to avoid (e.g. debt or household chores). For the purpose of this discussion, we will consider the case of dividing up some asset that everyone desires, but remember the converse case exists. The difficulty in fair division is that people might have distinct ideas of fairness; they might have some notion of a fair share that is different than others, and so the challenge is to ensure fairness across all players. We will discuss these variances to fair division, as well as analyze some fair division algorithms that assume some of them in the context of the problem.

2 Examples

The following examples illuminate the importance of fair division, as well as a few of the places it has been applied.

2.1 Cutting a Cake

You are at a potluck birthday party for your friend. Unfortunately, due to a total lack of coordination, you and all your friends brought only bottles of Brand Name Cola. Everyone came hungry, expecting a large feast, but the only food in sight is the birthday cake, which was provided by the host. The cake is a masterpiece, half chocolate and half yellow cake, and covered in icing and pieces of fresh fruit. There are strawberries, pineapples, raspberries, and blueberries, arranged in artful patterns. The cake is large enough to be cut so that there are pieces on the corner (with the most frosting but least fruit), pieces on the edge (with a lot of frosting and only some fruit), and pieces in the middle (not much frosting but tons of fruit). Everyone at the party wants to eat as much cake as possible. How do you decide to slice the cake, and how do you assign pieces, so that no guest feels like the red-headed stepchild who doesn't get enough cake?

2.2 Dividing an Estate

Mario and Daniel, although they are not related, find themselves as equal heirs to a modest estate. The estate includes a never-driven but two year old Audi A8, an enormous diamond fitted tastefully onto a necklace, and a full set of Griswold cast iron pans, with lids, which have been lovingly seasoned and cared for for a couple of generations. Mario and Daniel are friends, and would prefer if possible to leave this division feeling like they didn't get taken by the other. On the other hand, they don't even agree how much the Audi is worth. Daniel thinks the car is worth its full sticker value, but Mario insists it is worth less, because it is, after all, two years old. How can they divide the assets fairly? How does this change if the estate includes a large amount of cash? How does it change if there is an additional heir?

2.3 The Division of Germany following WWII

Aside from German reparations, demilitarization, and prosecution of war criminals mandated by the three Allied forces (US, USSR, UK) during the Yalta conference and Potsdam Agreement (1947), the quadripartite division of German occupation into annexing regions was a very involved process requiring much discussion. Although it is not publicly known if (say) a fair division algorithm or procedure was used to mediate the dividing of Germany, we can speculate as to how structured such governmental and geographical decisions were made among these Allied leaders. In the end, the outcomes of these divisions were in accordance and "fair".

Generally speaking, the players involved in the fair division process are advised to remain vigilant, since the aftermath claim of Germany and Poland allowed Joseph Stalin and the Soviet Union to claim eastern borders and create a stronger communist presence in Eastern Europe, which led to tensions between the US and the USSR at the start of the Cold War [1].

2.4 The Value of Fair Division

People have a tough time dividing assets. Families get torn apart over estate division arguments, and divorces are often incredibly messy. The settlement of a civil case, or negotiation of a corporate merger, can affect thousands of people in one single agreement. Because of this, a skilled mediator can make a very good living, just by helping people feel like a given division of assets is fair. Experienced courtroom mediators make six figures[1], and some may even make more.

[1] http://www.ncsconline.org/d_kis/jobdeda/Jobs_ADR_Counselor.htm

3 Basic Fair Division

In order to begin the discussion of fair division algorithms, it is necessary to establish the basic framing of the problem. The basic fair division problem is that a resource or good S must be divided among a set of players P such that all players receive a fair share. A fair share is whatever p makes of s, or more formally, $u_p(s)$ is the perceived value of s by p. Therefore, p's fair share constitutes $\frac{u_p(S)}{n}$ of S, where $|P| = n$ and where u is the utility function that performs a valuation of S for $p \in P$.Various approaches to this problem make several assumptions about S and P, as we will see. However, there are several invariants, or properties that these methods must assume in order for them to work correctly [2]:

1. The players are non-cooperative; the procedure must operate without "communication" between them.
2. The players act rationally, meaning that they act in their best interest and refrain from making emotional decisions.

The tip-off about "communication" implies that the players are not allowed to reveal their intentions or suggest choices to other players. These properties seem obvious, but in fact, they help maintain correctness ("fairness") of these procedures. In reading these extra properties and procedures, keep in mind how fairness is being evaluated and how it is being "computed" in a sense, as well as how the following variances define fairness in different ways.

3.1 Notions of Fairness

In order for a division to be fair, we must have a formal idea of fairness that can be applied. Unfortunately (or fortunately, perhaps), there are many potential measures.

Definition 1. *A division among n players is called **proportional** if each player's share is, by his own reckoning, at least $\frac{1}{n}$ of the total asset.*

There are many advantages to proportional division - in fact, proportional division is typically the easiest measure of fairness to understand, and the easiest to accomplish.

Definition 2. *A division is called **equitable** if all players agree that each player received an equal portion.*

An equitable division is a lot like a proportional division, but it is slightly more strict. Each player must think he received at least an equal portion, but additionally he must think every other player received an equal portion.

Definition 3. *A division is called **envy-free** if no player would prefer to have the share of another player, instead of the share he received.*

An envy-free division cannot always be achieved. Further, although it may be possible to achieve in practice, it is difficult to prove that any particular algorithm produces one.

Definition 4. *A division is called **optimal** if no player could have a larger portion of the asset without another player having less.*

An optimal division is basically one where no players envy each other's shares - no series of trades could make everyone happier than they already are with the division. In some texts [2], this is sometimes called a *Pareto optimal*. This measure alone isn't really a good guarantee of fairness, however - one person getting the whole asset and everybody else getting nothing is technically optimal.

3.2 Extra Variances of the Problem

Definition 5. *A good or resource S is called* ***continuously divisble*** *if S can be divided in a countable number of ways such that the value for S is not affected based on its division.*

Think of a cake or pizza; these are prime examples of a continuously divisble good.

Definition 6. *A good or resource S is called* ***discretely divisible*** *if S is not continuously divisble; in other words, the dividing of S may affect its value.*

Think of a set of goods such as an Audi A8 and some cast-iron pans. This set is considered the whole resource or good, and splitting the car in a continuous way may involve breaking up its mechanical parts, which will affect its value to the players. Instead, the players may resolve the division of a discretely divisible good by choosing which subset of goods to take, rather than their granular parts.

3.3 Number of Players

The number of players indeed may affect the outcome of the division such that one of these properties may break. To prevent this and uphold these assumptions, there are procedures (discussed below) that pre-determine the number of players involved in the fair division.

4 Fair Division Procedures

Here we explain several known fair division procedures, or algorithms that provide a fair division based on varying properties.

Running Time

Note that the running time of these algorithms are awkward to reason about; humans aren't thought of as computational entities with memory and storage limitations, and it is certainly true that the RAM model for assessing running times of algorithms based on the atomic operation would not extend smoothly to how humans perform work.

However, we can make a case for measuring computational time as the number of atomic operations that all players must perform without paying heed to how complex those operations are (to a certain extent). We will treat humans as agents of work and measure time and space complexity by the actions that they perform, such as valuing many pieces of S or randomly assigning a divider. For example, having each player perform a valuation on n pieces of S would be a quadratic time operation ($O(n * |P|) = O(n^2)$) when $|P| = n$. We will see such examples of agent operations when analyzing the following procedures.

4.1 Divide and Choose

The divide and choose proportional fair division method is the simplest one to examine and prove its "fairness". The procedure calls for exactly two players, say p_1 and p_2 and a good or resource S that must be continuously divisible (otherwise the process of splitting into equal halves may not be possible). The idea is this: randomly assign the roles for p_1 and p_2 to be the *divider* and *chooser*. The divider is the player that will split S into equal halves. The chooser, on the other hand, is the first player that will make a selection on the split pieces of S while leaving the remaining selection to the divider. The algorithm is stated below.

Algorithm 1 DIVIDE-CHOOSE-METHOD (P, S): The divide and choose procedure for splitting a continuously divisible good S into equal portions.

Require: Two players, p_1 and p_2, and a resource or good to be split S.
Flip a coin. If heads, ***divider*** $\leftarrow p_1$ and ***chooser*** $\leftarrow p_2$. If tails, reverse the roles.
divider cuts S into two halves equal in value.
chooser selects exactly one of the two pieces.
divider receives the remaining piece.
Ensure: Both p_1 and p_2 receive a fair share of S.

Proof of correctness and running time. To prove correctness, or "fairness" in this class of algorithms, we need to show that the good S is being split into fair shares in the eyes of both p_1 and p_2. Since the assignment of divider and chooser is random, it suffices to say that these complementary roles are being fairly chosen, and that there is an equal chance of any player to be a divider or chooser (indeed, 50% chance). We see that throughout the algorithm, the divider gets to perform a valuation on S and makes a fair-share division such that both pieces of S are equal in his eyes. This prevents the divider from "cheating", or in other words, making an inequitable split, since he will only receive one portion by the chooser's choice. Since the divider divides S, the chooser is able to effectively choose the piece she desires, which may or may not be an exact 50% value, but it is the maximum value out of both pieces guaranteed to be at least her fair share.

Therefore, it is easy to see that, once the divider divides and chooser chooses, both players will receive a fair share of S.

Note that this algorithm makes several guarantees: for one, it guarantees a proportional split, since there are two players, each of which recieve a fair share as proven above. It also guarantees envy-freeness, since the divider cannot possibly envy the chooser's share because he divided it in two equal valued halves. Also, the chooser cannot envy the divider's share because he chose the piece he wanted! Therefore, the algorithm is constructed in such a way so as to guarantee envy-freeness.

To prove the running time of this algorithm, we will examine the number of instructions that the players as agents must perform in order to render a fair division. Since $|P| = 2$ and S is of an arbitary size, we will say that the running time is independent of these variables, since it does not matter how big or small S is and the size of P is just a constant (2). The task of assigning divider and chooser is a simple coin flip taking constant time. Afterwards, the dividing of S is a single instruction for the divider and also takes constant time. Finally, the selecting of portions from both the chooser and divider also takes constant time. Therefore, the running time is simply $\Theta(4) = \mathbf{\Theta(1)}$. □

4.1.1 Example

4.2 Extension to n Players with The Lone Chooser Method

Divide-and-choose is by far the simplest means of splitting a resource when there are only 2 players. However, it can be modified to extend to further players, entirely on its own. This method is known as the lone chooser method [3], because at each step one player gets to choose until all players have chosen a share.

The lone chooser method starts out the same as divide-and-choose. One player divides the resource into two portions, and another player chooses which half he would prefer. Then both these players divide their shares into three portions, and a third player chooses one portion from each of the two players. Now the resource is divided (effectively) into fair thirds. If there are more players, each player splits his resource into

four portions, and the next chooser picks a piece from each person's pile, and so on. Pseudocode for this algorithm follows below.

Algorithm 2 LONE-CHOOSER-METHOD (P, S): The lone chooser method for n players and a good S.

Require: n players in P, and a resource or good to be split S.
  Randomly select a player $p \in P$.
  $D \leftarrow \{p\}$ // the set of dividers
  Give S to player p.
  Remove p from P.
  for i from 2 to n **do**
    Randomly select a player $p \in P$.
    for each player $d \in D$ **do**
      d divides his share into i equal portions.
      p selects and receives one of those portions.
      d receives the remaining portions.
    end for
    Add p to D.
    Remove p from P.
  end for
Ensure: All players in P receive a fair proportional share of S.

Proof of correctness. To prove this algorithm correct, we must ensure that every player receives a fair share.

The intial split, into halves, follows the standard divide-and-choose method, and therefore each player will have a fairly divided half of the resource. The base case is then taken care of.

Assume the division is fair among the first t players. That is, each has a fairly-divided $\frac{1}{t}$ portion of the total resource. Then, if when player $t+1$ chooses his portion the division is still fair, the algorithm is correct by induction. For player $t+1$ to choose, each of the first t players divides their share into $t+1$ portions. Since the best of these portions will be taken, the dividers will endeavor to make each of these portions equal in value. The resource is then divided into $t \cdot (t+1)$ equal portions. Player t will take t of these (one from each of the t dividers), leaving each divider with t portions. So every one of the $t+1$ players has t of the portions, which were divided up by the players to be as equal as possible. So the division of the resource among the $t+1$ players is also fair.

To discuss the running time of this algorithm, let's first assume that calculating the value of a share is an atomic operation, and that dividing a share into k portions take $O(k)$ time. Then, in this algorithm, we have a loop for $i = 2$ to n, and inside the loop we have another loop, which iterates $i-1$ times. In the inner-most loop, we have a division into i portions ($O(i)$ time) and i evaluations of those portions ($O(i)$ time again), and a constant time assignment of portions to people. In the outermost loop, we have a random selection ($O(1)$) and a few set assignments ($O(1)$), so overall we have $i-1$ loops of $O(i)$. This means, each time we run through our outer for loop, it takes $O(i^2)$ operations. Outside this loop, we have only a few assignments (constant time), so the overall running time of this algorithm is $\sum_{i=2}^{n} O(i^2) = \sum_{i=2}^{n} O(n^2) = O(n \cdot n^2) = O(n^3)$.

□

Although this division is fair, it is not guaranteed to be envy-free. Each new chooser must select one

portion from each player who has received goods; there is no option to choose two portions from the same player, even if those are the two most valuable portions. Further, the most valuable parts of the resource could quickly be taken from the initial dividers, and eventually be gathered together by a new chooser. In either of these situations, it is possible that some player will envy another player's share. However, each player is guaranteed that their share is fair.

4.3 The Lone Divider Method

Both the lone divider and the lone chooser methods work on more than just 2 players, but their procedures follow different workflows. The lone divider method can be construed as a recursive algorithm that involves n players, where there is only one divider and the other players as choosers, hence the name of the method. The basic workflow is as follows: the divider is randomly chosen, and he will divide the good S into n pieces, labeled $s_1, s_2, ...s_n$. Then, each chooser puts a value on each piece that they consider to be a fair share into a list. This is called a *bid* or *declaration* in some texts [2, 3]. Next, the bids are consolidated by an outside referee, or mediator that presents no influence or intervention to the players. For each bid examined, the resolution of values may be as simple as assigning a particular piece s_i to p_1, s_j to p_2, and so forth, where $i \neq j \neq k$.... However, if there is some contention where two or more players want the same piece, then a non-contested piece is assigned to the divider, and the procedure repeats, this time with S being split into $n-1$ pieces. If, however, only two players want a single piece, then the divider-chooser method is done to resolve the division. The process repeats until every player receives a fair share of S.

Algorithm 3 LONE-DIVIDER-METHOD (P, S): The lone divider method for n players and a good S.

Require: n players in P, an outside referee **referee**, and a resource or good to be split S.
  if $|P| = 2$ **then**
    Perform DIVIDE-CHOOSE-METHOD(P, S)
  end if
  Randomly assign a player as ***divider***.
  divider splits S into n pieces of equal value $\{s_1, s_2, ..., s_n\}$.
  for each player $p \in P-$ **divider do**
    for each piece $s \in S$ **do**
      p assigns a value to s of and places it in his/her bid.
    end for
  end for
  referee collects all $|P - 1|$ bids.
  if referee finds that all players value a unique piece $s \in S$ with no contest **then**
    referee assigns all fair shares in S to all players in P, including the divider.
  else if referee finds that two or more players want the same item and none other **then**
    referee assigns a non-contested piece to the divider.
    referee merges all remaining pieces of S together.
    Perform LONE-DIVIDER-METHOD($P-$ **divider**, S).
  end if
Ensure: All players in P receive a fair proportional share of S.

Proof of correctness. To prove this algorithm correct, we must ensure that every player receives a fair share.

Automatically, we can see that the divider will get his fair share upon splitting S into n pieces. To see this, notice what evaluations must be made once the outside referee collects all of the player-specific bids: in one (yet uncommon) case , all players have a unique piece they declare, in which case the referee assigns to each player his or her desired piece and gives the remaining one to the divider. In the other (more common) case, there is some contention; there are two or more players who desire the same piece. In this case, the referee assigns a non-contested piece to the divider, the remaining pieces of S are merged again, and the algorithm recurses, this time excluding the divider. Therefore, the divider is always guaranteed to receive his fair share, regardless of which condition occurs.

Having explained the algorithm in showing that the divider receives his fair share, it is now straightforward to show that all other players will receive one as well. We split our proof into case-by-case analysis: in the first case, it is easy to see that all players (minus the divider) are able to receive their fair share as noted on their bid. This is more like an algorithmic jackpot, in the sense that the outside referee was lucky to find that there was no contention and thus the algorithm did not have to recurse at all. All players receive their fair share, and the remaining piece goes to the divider, which we have already shown that he gets his fair share, too.

The second case is more complicated. In reality, it is very uncommon to find no contention of pieces of S among the players. In this case, the algorithm recurses, having satisfied the divider and excluding him from the next iteration. However, we see that a new divider must be chosen, and thus he must be satisfied as well. But just as we have shown that one divider gets his fair share for a single iteration of the algorithm, we can see that any succeeding divider will also get his fair share, regardless of how many players are left. Indeed, when there are $|P| > 2$ players left, the divider will get his fair share. When $|P| \leq 2$, then we delegate the rest of the task to the DIVIDE-CHOOSE-METHOD, which guarantees a fair share for both a single divider and chooser.

Therefore, we have shown that, for n players, each player, at some point becoming a divider or being sent to perform the DIVIDE-CHOOSE-METHOD, will receive his fair share of S.

Now we will attempt to prove the running time for this algorithm. Taking a walk through it, we see a few tasks occur sequentially, namely: The divider is randomly assigned out of n players, which we can view as taking constant time ($O(1)$); the divider splits S into n pieces of conceivably equal value, which can be thought of as a linear time operation ($O(n)$); for each player minus the divider, and for each piece $s \in S$, the player assigns a value to s and places it in his/her bid, which looks like a quadratic-time operation ($\Theta((n-1) * n) = O(n^2)$); The referee collects all of the bids and evaluates them, which again must be a cross-comparison of pieces of S to players in P, a quadratic-time operation ($O(n^2)$); Finally, the referee must do some linear amount of work to assign all fair shares in S to all players, or he must assign a non-contested piece to the divider and merge all remaining pieces of S together, both of which are also linear-time ($O(n)$) operations.

Having walked through this, we see that the leading term signifies quadratic work as a function of growth w.r.t. n. However, we also see this as a recurrence relation, since this is in fact a recursive function. This function resembles a recurrence like the following:

$$T(n) = T(n-1) + 2 * O(n^2) + 2 * O(n) + O(1) \tag{1}$$

Having shown this, we can see that the recurrence performs a head recursion that produces a recursive call tree of depth $n-1$, each performing some quadratic work. Therefore, we can confidently say that this algorithm produces a leading term of $O(n^3)$, since there are $n-1$ calls for work containing quadratic ($O(n^2)$) terms, thus producing a running time of $O(n^3)$. Note that this algorithm yields the same running time complexity as the LONE-CHOOSER() procedure, yet its workflow is quite different.

□

4.4 The Moving Knife Method

The "moving knife" algorithm can be applied in the case of a continuously-divisible good. It gets its name from the classic cake-cutting example problem, and works similarly to the lone divider method. However, the algorithm itself is continuous. In this case, a referee is responsTible for cutting the cake. He does so by moving the knife slowly, and continuously, across the surface of the cake. As soon as any player believes the portion to be sliced off the cake is a fair share, that player says so. The cake is cut at that point, and the slice is given to the person who asked for it. The process is repeated on the remainder of the cake. Of course, S does not have to actually be a cake, but this motivates the naming convention for this algorithm. This algorithm also guarantees a fair division.

Algorithm 4 MOVING-KNIFE-METHOD (P, S): The moving knife method for n players and a continuously divisible good S.

Require: n players in P, an outside referee **referee**, and a resource or good to be split S.
  while S has not been fully assigned **do**
    portion $\leftarrow \emptyset$
    if any player $p \in P$ requests **portion then**
      Give **portion** to p and remove p from P.
    else
      slightly increase **portion** and remove the increase from S.
    end if
  end while
Ensure: All players in P receive a fair proportional share of S.

Proof of correctness. To prove this algorithm correct, we must ensure that every player receives a fair share.

Any player who receives a portion must receive a portion that he thinks is fair - otherwise, he shouldn't have requested that portion. The tricky bit here is proving that no player receives less than a fair portion.

Consider a player p who receives no portion, or else a portion which he believes is less than a fair share. This player cannot be one who requested a portion, because requesting a portion is a signal that the portion is fair. So, this player was left with the remainder after all other players had requested their portions. There is only one way this could happen - namely, that some or all of those players received more of S than was fair. Let q be a player who received more than their fair share. Since the "knife" is moved continuously, there was a point at which the cut would have been fair. Since q received a greater-than-fair share, q waited until the portion was fair, at which point p did not request it, and then it increased even further, and then q requested it. So, if p honestly believed the portion was fair at some intermediate point, p should have requested it.

p cannot end up with less than a fair amount unless q has a greater-than-fair share. q cannot have a greater-than-fair share unless p is holding out for an *even*-greater-than-fair amount of S, so it is in the best interest of p to request the first fair share available. And, acting rationally (by assumption), p will do so. So, according to the assumptions of the players in the division, the division will be fair.

□

4.5 Dealing with Discretely Divisible Goods - Sealed Bids

So far, we have touched on several fair division schemes that are proportional, envy-free, and have 2 or more players involved in them. These schemes assumed that the good S was continuously divisble, or that S can be divided up in a countable number of ways such that its value is not affected based on its division. Now we will look at an algorithm called the "Sealed Bids" algorithm that describes an approach for fairly dividing S that is not continuously divisible but *discretely* divisible.

We start with a motivating example. Suppose that a family has gathered at a meeting to discuss the inheritance of its late patriarch who passed away several weeks ago. He left behind his home estate with several goods including his old mint-condition Chevrolet, collection of award-winning fishing rods, and his enormous family portrait drawn by a famous artist. These things, including his home, are being divided over by the n family members who assigned an outside but non-intervening mediator to lead and operate the division properly using the "sealed bids" method. These family members have enough money to propose high bids for these desired items, as we will describe next.

The sealed bids method [3, 4] takes as input a discretely divisible good S and a set of players with enough money P and allows players to privately give honest bids for each item in S. This time, however, they will be attaching a dollar value to each item as a bid and must have enough money to place a bid. Note that a player's fair share is exactly $1/n$ of his or her total assessment. Just as in the lone-divider method, an outside referee or mediator will collect these bids and control the division as an agent in the algorithm. Specifically, for each item in S, the mediator will assign it to the highest bidder. This means that any high bidder may be able to afford multiple items from S. Therefore, the mediator must evaluate if the highest bidder's assessed value of the items received exceeds her fair share, she must pay the difference. On the other hand, if the assessed value of the items received falls short of a fair share, then she is paid out of money that other players have had to pay. Finally, once each item has been assigned to a player, the mediator will take any surplus of money and divide it equally among the players. Below is the psuedo-code for an algorithm resembling the sealed bids method:

Although this method seems to produce a fair share for all players, it has its limitations [3]. For one, all players must be able to put a "price tag" on all items in S, because if not, then one player with a large sum of money could potentially bid the highest for all items, and then the paid-out players would not receive their fair share. There really are no *favorite* items that players can hold on to; all players must be able to put a price on any item.

Additionally, in this kind of problem motivated by our early example, the players might very well know each other's intentions. For example, player p_1 might place a higher bid than player p_2 on s knowing that p_2 will place all of her money on it. And perhaps p_2 knows this fact, too, so how can she possibly afford to win s? The problem with presupposed or pre-existing knowledge of player may not affect fair shares but does affect desired outcomes, or "best" shares. In other words, this method definitely does not guarantee an envy-free outcome.

Proof. We will show that this algorithm produces a proportional fair-share for all players.

The algorithm begins with a double-for loop on the players $p \in P$ and then the items $s \in S$. For each s and p, the player p will assign a dollar value on s for the bid. Once each p has done this, then he or she will calculate the fair-share based off of these values in the bid. Our aim is to show that this fair-share has been met for each player.

We can see that the referee gives the item to the highest bidder, so let's suppose that a player p simply puts all of his money into an item s that has been taken by a higher bidder. Then, of course, p must receive

Algorithm 5 SEALED-BIDS-METHOD (P, S): method for dividing a discretely divisible good S.

Require: n players in P, an outside **referee**, and a discretely divisible resource or good to be split S.

for each $p \in P$ **do**
  for each item $s \in S$ **do**
    p places a dollar value on s for the sealed bid.
  end for
  p calculates fair share based on values in bid.
end for
referee collects all bids from all players.
for each item $s \in S$ **do**
  referee determines highest bidder based upon maximum dollar value on s.
  if more than one player bids the highest bid **then**
    Resolve tie in a pre-determined fashion.
  end if
  referee assigns s to the highest bidder for s.
end for
for each player $p \in P$ **do**
  if values of all received items for $p > p$'s fair share **then**
    p pays the difference between the fair share and values of received items to the holding pile.
  else if values of all received items for $p < p$'s fair share **then**
    p gets paid the difference between the fair share and values of received items from the holding pile.
  end if
end for
for each player $p \in P$ **do**
  p receives $1/n$ of holding pile for final allocation.
end for

Ensure: All players in P receive a fair proportional share of S.

his dollar value on s by the end of the procedure, otherwise he will not receive his fair share. According to the algorithm, this is definitely the case and occurs once all of the high bidders receive their items besides p. When the algorithm iterates to evaluate the value of all items received by each player, we will see that p has no value for no items received, and so he must be paid the value he placed on s from the holding pile. We can see that all players will eventually receive their fair share, whether it be from the items bidded on, or from the difference between a calculated fair share and the value of items received from each player.

The running time for this algorithm must pay attention to $|S|$, since it is now known that $|S|$ does not equal $|P|$. Walking through the procedure, we see that, for each player $p \in P$ and $s \in P$, p must valuate s. This is a $O(|P||S|)$ time operation. Moving forward, there are some linear-time loops for the referee to determine the highest bidder ($O(|S|)$), for each player to determine if his value of all received items is greater or less than his fair share ($O(|P|)$), and for each player to receive $\frac{1}{n}$ of the holding pile (also $O(|P|)$). Together, the running time is subsumed by the earliest operation, namely $O(|P||S|)$, and so the running time of the sealed bids algorithm with a discretely divisible S is $O(|P||S|)$. □

5 Sperner's Lemma

Now we move in a much different direction, and discuss a lemma from discrete mathematics. It can be used to guarantee an envy-free division of a continuous resource, so it's incredibly useful. It's called Sperner's Lemma, and it deals with tesselations of n-simplices. First, I'll define these terms.

Definition 7. *An n-simplex is a triangulation of $n+1$ points. For instance, a 1-simplex is a line segment. A 2-simplex is a triangle. A 3-simplex is a tetrahedron, and so on.*

Definition 8. *A tesselation of an n-simplex is a division of the simplex into smaller n-simplices, which together fill up the entire original simplex. The smallest unit of these divisions are called elementary simplices.*

Figure 1 shows tesselated n-simplices for $n = 1, 2$, and 3.

Sperner's lemma deals with labeled tesselations of n-simplices. A labeling of an n-simplex is assigning a label to each corner of each elementary n-simplex. So, a labeling of a 1-simplex (line segment, divided into sections) is simply a labeling of each dividing point and each end point.

A Sperner labeling of an n-simplex satisfies a few properties. The corner points of the overall n-simplex have unique labels, say 1 through $n+1$. The points on an edge between two corners are labelled only with the labels of the corners (the line between 1 and 2 only contains labels 1 and 2). Further, a face of the n-simplex with three corners contains only the labels of the corners, and so on. The interior points can have any labeling.

Figure 2 shows a Sperner labeling of a 1-simplex and a 2-simplex. We now have all the tools needed to state Sperner's lemma.

Lemma 1 (Sperner's Lemma)**.** *For any Sperner labeling of an n-simplex, there are an odd number of elementary n-simplices with all labels. In particular, there is at least one.*

Before we prove this, I'd like to make a quick remark about how we will use this for division. If each vertex on an elementary simplex is associated with a player, and associated with a particular division of S, then we can get a Sperner labeling of the simplex by asking the player which piece of S he would prefer. This will lead us to a division where each player prefers a different piece.

Proof. (By induction on n.) To prove this, let us first consider the base case of a 1-simplex. A Sperner labeling on a 1-simplex is simply a labeling where one end of the line segment is labeled "1", the other end

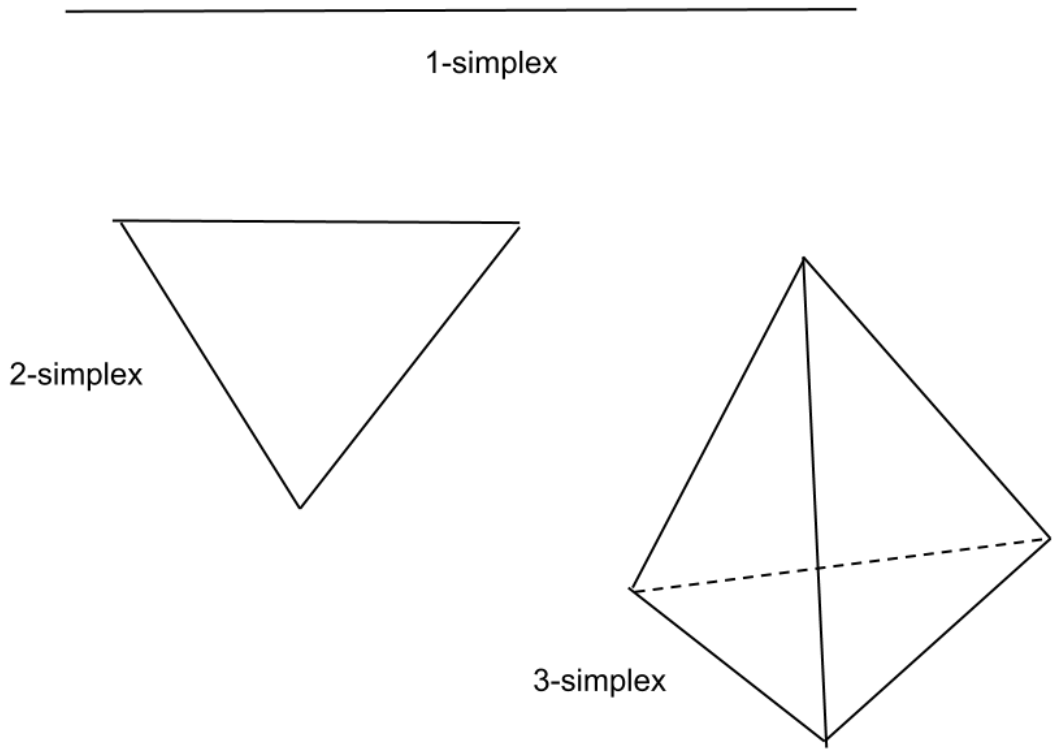

Figure 1: A Few n-simplex Examples

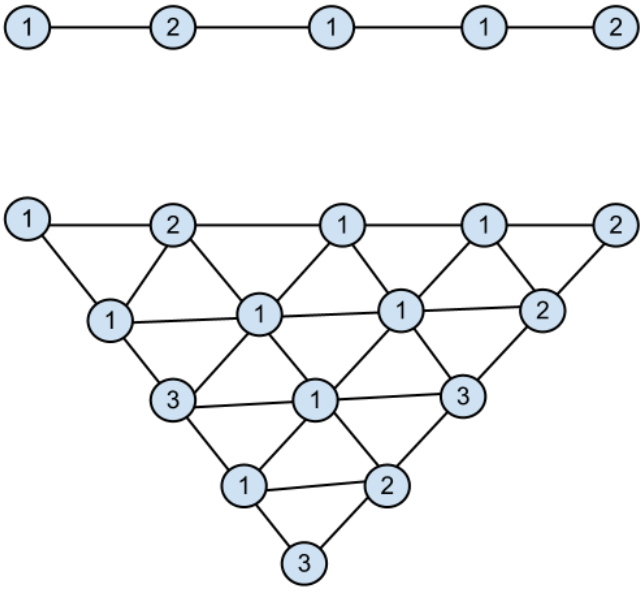

Figure 2: Sperner Labelings on Small n-simplices

is labeled "2", and all intermediate points are labeled as one of these options. Sperner's lemma states that this simplex must have an odd-number of elementary segments with both labels. Because we start at "1" and end at "2", we must have at least one such segment - the only way to switch labels is to switch on some elementary segment. Further, if we had an even number of elementary line segments with both labels, they would pair off - we would switch from "1" to "2," and then back to "1." So, we must have an odd number of fully-labeled elementary simplices, and Sperner's lemma holds for our base case.

Now we move on to the inductive step. Assume Sperner's lemma holds for $n-1$. That is, a Sperner labeling of an $n-1$ simplex contains an odd number of fully-labeled elementary simplices. Turning an $(n-1)$-simplex into an n-simplex simply requires adding a new point, and carrying out the triangulation from all prior corners to the new point. So, an n-simplex has n faces, each of which is an $(n-1)$-simplex. A Sperner labeling of the n-simplex requires that the face between the original corners is itself a Sperner-labeled $(n-1)$-simplex. This is how we will use the inductive hypothesis.

Consider the n-simplex as a house, and call each elementary n-simplex a room in that house. Call any face of an elementary n-simplex a door if it contains all labels $1, 2, \ldots, n$. Remember, there are $n+1$ distinct labels, and each room has $n+1$ corners. So, any room may have zero doors, one door, or two doors. The reasoning behind this is as follows. A room with a door contains n distinct labels, and has one additional corner. If that corner is a repeat of one of these labels, then the room has exactly two doors: the first door considered, and the door which uses the repeat-labeled corner. If that corner is not a repeat, then it must be labeled $n+1$, and so the room has only one door. Further, a room with only one door is a fully-labeled elementary simplex. Of course, not all rooms need to have doors, but a room with no doors cannot be a fully-labeled elementary simplex, so we need not consider it.

One face of the house is a Sperner-labeled $(n-1)$-simplex, and by the inductive hypothesis contains an odd number of fully-labeled elementary $(n-1)$-simplices. However, a fully-labeled elementary $(n-1)$-simplex is the same thing as a door. So, the Sperner-labeled n-simplex has an odd number of doors from its face. To find fully-labeled elementary n-simplices, we simply walk through these doors. Each door leads us in to a room, where there is either another door (we'll call it a trapdoor), or there is not. If there is no other door, we have found a fully-labeled elementary n-simplex. If there is a trapdoor, we walk through it. Once again, we're in a new room, and we'll continue walking through trapdoors until we find an elementary n-simplex. There is no chance we'll double-back on ourselves, because no room has more than two doors.

There is, of course, another possibility. We may be led along a path which leads back out the $(n-1)$-simplex face. However, in this case, two of the doors on that face pair off, leading only to one another. Since there are an odd number of doors on this face, taking away a pair of doors still leaves an odd number of doors which lead to fully-labeled rooms.

There may be other fully-labeled rooms in the interior of the house, unaccessible from the $(n-1)$-simplex face. These rooms must have doors, and since there are no doors on any other faces (because it is a Sperner labeling), the doors must lead from one interior room to another, in pairs. So, we have an odd number of fully-labeled rooms from doors on the surface of the simplex, and an even number of additional rooms, for a total of an odd number of fully-labeled rooms in our n-simplex.

An example showing all these possibilities is presented in Figure 3. Although the example is a 2-simplex, the argument works for general n. □

So, now we have Sperner's lemma, which means with a Sperner-labeled, tesselated n-simplex, we can find an elementary simplex containing all labels. How do we leverage this into an envy-free division?

First, note that dividing a continuously-divisible S across n players requires us to divide it into n sections. The sum of these sections is S. So, if each piece of the resource is called x_i, for any particular

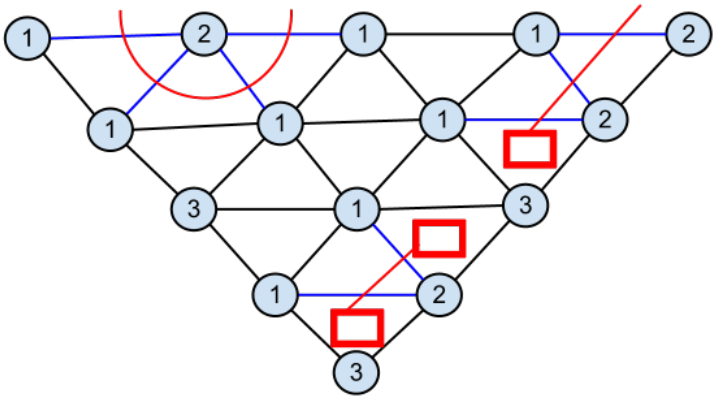

Figure 3: A Sperner-labeled 2-simplex. "Doors" are marked in blue, paths are marked in red. Elementary simplices with all labels are marked with a red square.

labeling, we have $x_1 + x_2 + ... + x_n = 1$, and each $x_i \geq 0$. This describes a region in $n+1$-dimensional space.

We'll break it down a little bit. Consider two players, dividing a resource into pieces x_1 and x_2. Then $x_1 + x_2 = 1$ is a line from $(1, 0)$ to $(0, 1)$. Any point along that line is a particular division. With three players, $x_1 + x_2 + x_3 = 1$ is a triangle in the first octant, with corners $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. Any point along that triangular region is a particular division of S. By now the pattern should be apparent. Dividing a good among n players describes an $(n-1)$-simplex, where any point inside the simplex represents a division.

Let's define a couple terms, and then we can formalize this into a theorem. The proof will work by construction, and demonstrate how to find an envy-free division.

Definition 9. *A player is called hungry if he would prefer a non-empty portion of S to any empty portion. This term comes from the cake-cutting problem: if you're hungry, any cake is better than no cake.*

Definition 10. *A player's preference set is a mapping from a division to a portion, stating which portion the player would prefer given that division. A player's preference set is closed if for any set of divisions converging to some particular division, if the player's preference remains the same on each of those divisions, it remains the same at the particular division.*

The closed preference set may seem a little funky, but basically it just means that if your preferences don't change randomly as the division gets finer.

Theorem 1. *For hungry players with closed preference sets, there exists an envy-free division of a continuously divisible good S.*

Proof. Above we discussed how a division of S into n pieces describes an $(n-1)$-simplex region. We can tesselate this region, so that each point represents a particular division. Associate each point with a player, such that each elementary simplex has one corner associated with each player. This can be done if the tesselation is barycentric; see [5]. For each point, ask the associated player which portion he would choose, given that division. The player will give a number 1 to n. Label the point with this number.

The labeling of this $(n-1)$-simplex is a Sperner labeling. This is true, because each corner represents a division where only one piece is non-zero: hungry players all prefer the non-zero piece, and so no matter which player is associated with a corner, the corner labels will be 1 through n. Along the edge between labels i and j, the only non-empty pieces are i and j, so no other label will appear. Along any face of the figure, there is some empty piece in the division. Since nobody will choose the empty piece, that label will not appear on that face. So, hungry players will automatically produce a Sperner labeling.

By Sperner's lemma, there is at least one elementary simplex containing all labels. Since each elementary simplex is labeled, each corner by a different person, we have a set of divisions where each player prefers a different piece. Therefore, we have found a region where an envy-free division is possible.

If any of these divisions works out that each player prefers a different piece, that is an envy-free division and we are finished. If not, we simply use a finer tesselation mesh in this region and repeat. Because we keep "zooming in" on a region, we eventually converge to a particular division. Because players have closed preference sets, the particular division we approach is envy-free. □

For additional reading on Sperner's lemma, and a perhaps more clear description of using Sperner's lemma for fair division, see [5]. It is the primary reference for all the Sperner's lemma material in these notes.

References

[1] The Cold War; Fair Division; Proportional Fair Division; The Potsdam Agreement; German Territory after WWII. Wikipedia.

[2] Lippman, David. Fair Division. David Lippman Home Page. http://dlippman.imathas.com/mathinsociety/FairDivision1.3.pdf

[3] Peressini, Dominic. Fair Division Problems and Fair Division Schemes. The Discrete Mathematics Project. University of Colorado at Boulder. January 1997. Retrieved from http://www.colorado.edu/education/DMP/fair_division.html

[4] Bowen, Larry. Math 103: Three types of Fair Division. University of Alabama Center for Academic Success. Retrieved from http://www.ctl.ua.edu/math103/fairdiv/typesof.htm

[5] Su, F. E. (1999). Rental harmony: sperners lemma in fair division. America, 106, 930-942.