Spliddit: Unleashing Fair Division Algorithms

Jonathan Goldman

Advisor: Ariel Procaccia

Abstract

The field of fair division has been rapidly expanding in recent years, capturing the interest of researchers in economics, mathematics, and computer science. The literature encompasses provably fair solutions for a wide variety of problems — many of them relevant to society at large. However, few fair division systems are used in practice, with even fever available to the public. Enter Spliddit, a first-of-its-kind website which provides easy access to carefully designed methods for dividing rent, goods, credit, chores, and fares. Since launching in November 2014, Spliddit has received coverage in popular technology websites such as Gizmodo and Fast Company, and has been used by tens of thousands of people. In this thesis, we'll discuss the overall design and implementation of Spliddit, as well as the algorithmic details of Spliddit's various applications.

1 Introduction

Fair division theory has evolved into a major field of study in mathematics, computer science, and economics over the past several decades. Historically, much of the literature has focused on designing and analyzing various notions of fairness. Nowadays, more attention is being devoted to developing *provably fair* solutions to a wide variety of problems. However, despite the relevance of many of these problems to society at large, very few fair division methods have been made publicly available. Exceptions include the Adjusted Winner Website, which provides access to a method for dividing indivisible goods between two people, and Francis Su's Fair Division Calculator, which implements methods for splitting rent, divisible goods, and chores. However, these websites are both limited in scope and usability. Other applications, including Splitwise and Splittable, offer functionality for splitting bills but rely on simple methods which don't offer mathematical fairness guarantees[1].

Enter Spliddit (`www.spliddit.org`), a new fair division website. Quoting from the website:

> *Spliddit is a not-for-profit academic endeavor. Its mission is twofold:*
>
> - *To provide easy access to carefully designed fair division methods, thereby making the world a bit fairer.*
> - *To communicate to the public the beauty and value of theoretical research in computer science, mathematics, and economics, from an unusual perspective.*

Spliddit launched on November 4, 2014 with applications for sharing rent, dividing goods, and assigning credit. On April 28, 2015 Spliddit launched additional applications for splitting taxi fare and distributing tasks. Spliddit has attracted tens of thousands of unique visitors, building on press coverage in popular technology and science websites such as Gizmodo, Lifehacker, and Fast Company. In this thesis, we'll discuss the design

[1]The websites mentioned in this section are `www.nyu.edu/projects/adjustedwinner/`, `www.math.hmc.edu/~su/fairdivision/calc/`, `www.splitwise.com`, and `www.splittable.co`.

and implementation of Spliddit. Section 2 details many of the important design decisions made and how they relate to the mission statement quoted above. Then, section 3 delves into the five different applications, including the mathematical models, fairness guarantees, and algorithms, and section 4 discusses some technical details regarding Spliddit's implementation. To demonstrate the impact of Spliddit and in particular how well Spliddit achieves its mission, section 5 gives some statistics and testimonials. Finally, section 6 concludes by exploring Spliddit's potential to serve as a platform for empirical fair division research.

2 Design and Ideology

Before diving into Spliddit's implementation, it is worth taking a look at some key design decisions, and in particular how they contribute towards Spliddit's mission.

In order to be useful to society at large, Spliddit's first three applications were chosen to maximize broad appeal and usability. The Sharing Rent application, which helps people moving into a new apartment with roommates to fairly assign rooms and split the rent, can be used by some 32.0% of American adults who live with roommates[2]. The Dividing Goods application may be useful to the 800,000 couples a year in the United States who seek divorce[3], but is also useful for estate planning and any other situation in which a collection of goods are to be shared among multiple people. Finally, the Assigning Credit application may be used to help determine contributions of individuals to any group project, such as a research paper, school project, or business endeavor. The remaining two applications were chosen based on user feedback: the Distributing Tasks application is designed for divvying up household chores or work shifts, and the Splitting Fare application fairly divides the cost of a taxi ride between friends who need to be dropped off in different locations. In summary, each of the five applications apply to common problems faced by millions of people. Figure 1 shows the five applications as they appear on Spliddit's homepage.

[2]Roommate statistic from Zillow's "Doubled-up for Dollars" study: `www.zillow.com/research/doubling-up-households-7947/`.

[3]Divorce statistic from `www.cdc.gov/nchs/mardiv.htm`

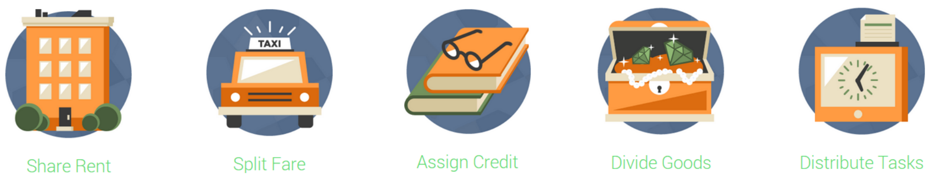
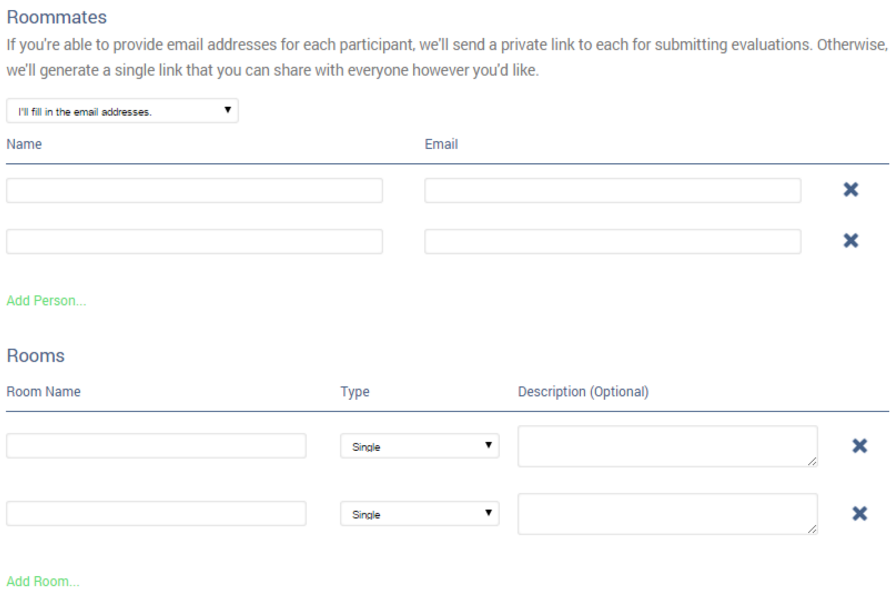


Figure 1: Spliddit's five applications.

Ease-of-use is another key aspect of Spliddit's ideology. Each of Spliddit's applications have three phases. In the first phase, a user specifies all of the *inputs*: the list of participants, the list of resources (rooms, goods, tasks, etc.), and any remaining information needed to describe the problem (for example, the monthly rent.) Figure 2 illustrates the first phase of the Sharing Rent application.

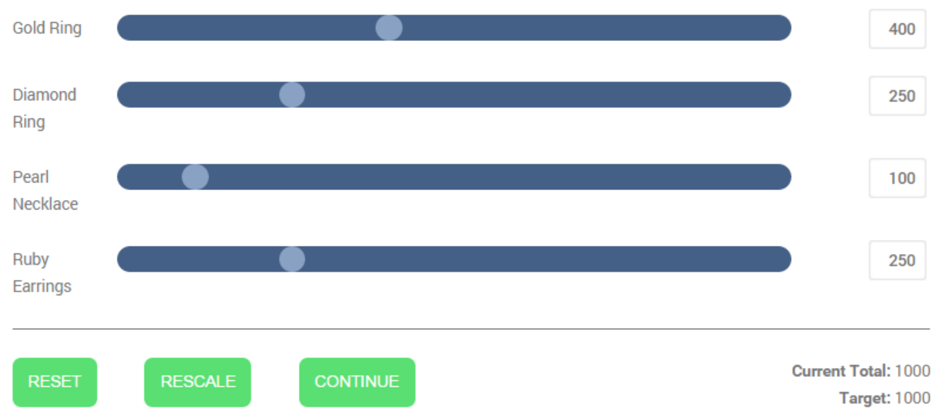


Roommates

If you're able to provide email addresses for each participant, we'll send a private link to each for submitting evaluations. Otherwise, we'll generate a single link that you can share with everyone however you'd like.

Figure 2: The interface for creating a new Sharing Rent instance.

In the second phase, each participant enters her *evaluations*, which indicate preferences over the resources. In many fair division models, preference elicitation can be quite complex: for example, Su's rent division algorithm requires participants to take turns answering a long series of questions of the form "which room do you prefer at this price vector?" [3]. This is problematic because it is time consuming and requires all participants to be in the same room at the same time. In contrast, all of Spliddit's applications are designed to require minimal input from the user, and to allow users to submit their evaluations independently of one another. Figure 3 illustrates the evaluations form for the Dividing Goods application, in which users are asked to divide a pool of 1000 points among the items. It is important to note that simplifying user preferences typically requires making some additional assumptions. For the Dividing Goods and Distributing Tasks applications, we assume that preferences are *additive*, i.e. the value a user derives from receiving two goods is the sum of values derived from each individual good. For the Sharing Rent application, we assume *quasi-linear utilities*: each housemate wishes to maximize the difference between her value for a room and its price. These assumptions, which are clearly indicated on the website, are common in the literature and thus we believe including them to be a worthwhile tradeoff.

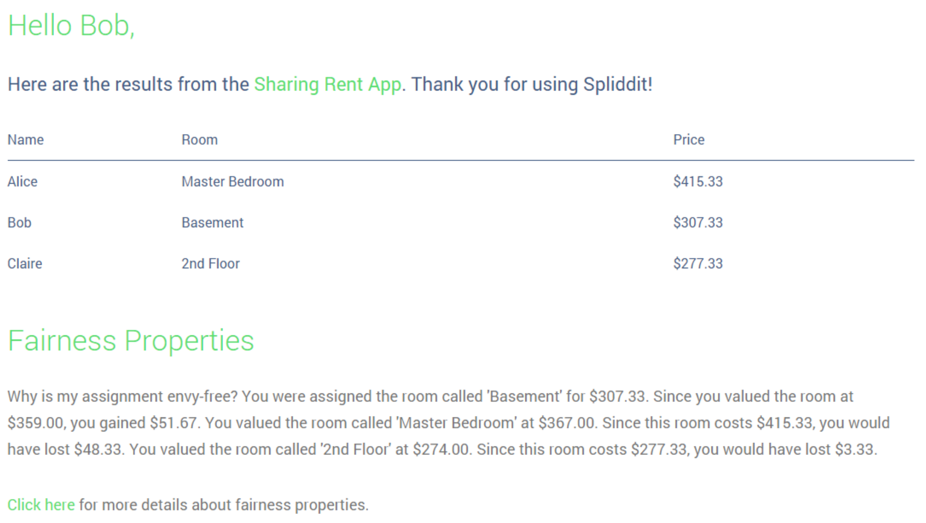


Gold Ring	400
Diamond Ring	250
Pearl Necklace	100
Ruby Earrings	250

Current Total: 1000
Target: 1000

Figure 3: The bidding interface for the Dividing Goods application.

In the third and final stage, the solution is presented, as illustrated in Figure 4 for the Sharing Rent application. Spliddit uses email to coordinate the three phases: when a new instance is created Spliddit sends private evaluation links to each participant, and once all evaluations are submitted Spliddit sends emails to each participants with the results. This stage also includes personalized information explaining why the presented solution is mathematically fair.



Hello Bob,

Here are the results from the Sharing Rent App. Thank you for using Spliddit!

Name	Room	Price
Alice	Master Bedroom	$415.33
Bob	Basement	$307.33
Claire	2nd Floor	$277.33

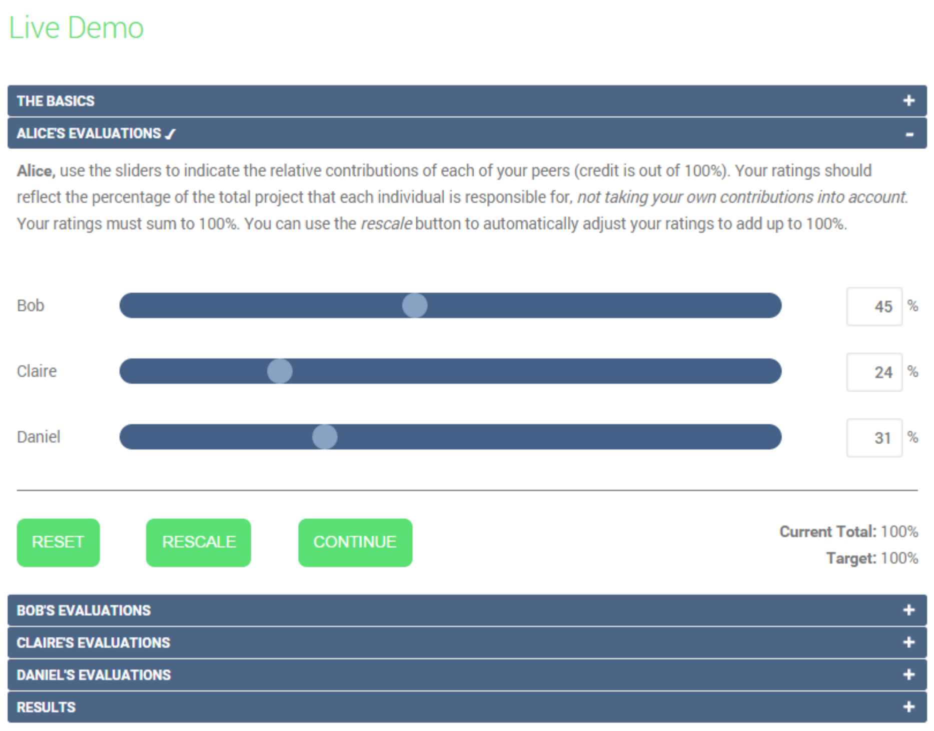
Fairness Properties

Why is my assignment envy-free? You were assigned the room called 'Basement' for $307.33. Since you valued the room at $359.00, you gained $51.67. You valued the room called 'Master Bedroom' at $367.00. Since this room costs $415.33, you would have lost $48.33. You valued the room called '2nd Floor' at $274.00. Since this room costs $277.33, you would have lost $3.33.

Click here for more details about fairness properties.

Figure 4: The results interface for an example run of the Sharing Rent application.

For convenience, and as a response to heavy demand, we added a *live demo* mode in which users can quickly create application instances, place bids, and compute the results on a single screen. This is especially helpful for new users who wish to obtain a better understanding of how altering bids can affect the outcome. The demo for Assigning Credit is illustrated in Figure 5.



Live Demo

Alice, use the sliders to indicate the relative contributions of each of your peers (credit is out of 100%). Your ratings should reflect the percentage of the total project that each individual is responsible for, *not taking your own contributions into account.* Your ratings must sum to 100%. You can use the *rescale* button to automatically adjust your ratings to add up to 100%.

Bob	45 %
Claire	24 %
Daniel	31 %

Current Total: 100%
Target: 100%

Figure 5: The demo mode for the Assigning Credit application.

Another major aspect of Spliddit's ideology is to convey the importance and beauty of mathematics and computer science, specifically as applied to fair division theory. All of the methods implemented by Spliddit yield *provably fair* solutions, using fairness notions which can be formalized mathematically. The landing pages include descriptions of the fairness guarantees, animations illustrating these guarantees, a high level overview of the algorithms used, and links where one could further explore these algorithms. Some screenshots of the animations are shown in Figure 6. In addition, as previously mentioned we include personalized explanations for why our allocations are fair when presenting results (see Figure 4 for an example.)

Figure 6: Some screenshots from the animations included on the various landing pages used to illustrate fairness concepts. Each of these concepts are formally defined in Section 3.

3 Fair Division Models and Algorithms

We're now ready to formally describe each of Spliddit's applications, along with their corresponding algorithms and fairness guarantees. A single algorithm is provided for each application, even in cases where several incomparable approaches are present in the fair division literature. This nontrivial design choice is driven by usability: it allows us to focus on giving an accessible explanation (enhanced by animations) of the guaranteed fairness properties in the context of each of the applications. It's also worth noting that while algorithm runtime is important for being able to serve a large number of users quickly, the primary consideration is always to provide strong fairness guarantees.

3.1 Sharing Rent

In the rent division problem, we have a set of agents $I = \{1, 2, \dots, n\}$, a set of rooms $R = \{1, 2, \dots, n\}$, and the monthly rent $P \in \mathbb{R}_+$. We assume that each player i has nonnegative value v_{ij} for room j so that for all i, $\sum_j v_{ij} = P$. We further assume *quasi-linear* preferences, that is, if agent i receives room j at price p_j, $i's$ utility is $v_{ij} - p_j$. A solution is a pair (σ, p) where σ is a bijection from I to R, and $p \in \mathbb{R}^n$ satisfies $\sum_i p_i = P$. We say a solution (σ, p) is *envy-free* if for each $i, j \in I$, $v_{i\sigma(i)} - p_{\sigma(i)} \geq v_{i\sigma(j)} - p_{\sigma(j)}$. It is straightforward to show that in this domain, all envy-free solutions are also *Pareto efficient*, that is, there is no other allocation for which all players have weakly greater utility, and at least one player has strictly greater utility. In fact, we can prove something slightly stronger: if (σ, p) is envy-free, than σ is a *socially optimal* assignment in that it maximizes the sum of utilities across all players. In symbols, σ is socially optimal if $\sigma \in \text{argmax}_{\sigma'} \sum_i v_{i\sigma'(i)}$.

Lemma 1. *If (σ, p) is envy-free, than σ is socially optimal.*

Proof. Suppose (σ, p) is envy-free and let σ' be any bijection from I to R. By envy-freeness of (σ, p), for each $i \in I$, $v_{i\sigma(i)} - p_{\sigma(i)} \geq v_{i\sigma'(i)} - p_{\sigma'(i)}$. So, summing all of these inequalities, we obtain

$$\sum_{i \in I} (v_{i\sigma(i)} - p_{\sigma(i)}) \geq \sum_{i \in I} (v_{i\sigma'(i)} - p'_{\sigma'(i)}). \tag{1}$$

Since σ and σ' are bijections and the components of p sum to P, adding P to both sides yields

$$\sum_{i \in I} v_{i\sigma(i)} \geq \sum_{i \in I} v_{i\sigma'(i)}. \tag{2}$$

□

Moreover, Klijn shows that for every socially optimal σ, there exists p such that (σ, p) is envy-free [1]. However, this p is not guaranteed to be positive. In fact, Brams and Kilgour observe that there are situations where some prices are negative at every envy-free allocation — some players are paid to live in the house [2]! Since it is unlikely that housemates would agree to such an arrangement, a natural question arises: when do nonnegative envy-free allocations exist? Su's Rental Harmony Theorem [3] provides a partial answer to this question. In the case of quasi-linear preferences, Su shows that if each person always prefers a free room to the most expensive room at any nonnegative price vector p, then there exists an envy-free solution[4]. Equivalently, as long as no agent values one room $\frac{P}{n-1}$ more than another room, nonnegative envy-free prices are guaranteed to exist. This condition seems likely to be met in practice, where rooms of an apartment are rarely drastically different from one another.

With this in mind, we seek an algorithm which yields envy-free solutions at nonnegative prices whenever possible. The following result due to Gal et al. gives us one possible approach.

Lemma 2 (Gal et al. Lemma 2.6)**.** *Suppose (σ, p) is envy-free, and σ' is socially optimal. Then, there exists a p' such that for all $i \in I$, $v_{i\sigma(i)} - p_{\sigma(i)} = v_{i\sigma'(i)} - p'_{\sigma'(i)}$, where p' contains the same values as p, possibly in a different order.*

Proof. See [4]. $\square$

Using this result, we conclude that if there exists an envy-free allocation at nonnegative prices, than for every socially optimal σ there exists a nonnegative p such that (σ, p) is envy-free. So, a candidate algorithm would simply identify a socially optimal allocation and then solve for envy-free prices via a linear program.

Despite its prevalence in the literature, envy-freeness is often not enough to satisfy our intuitive notions of fairness. For example, consider the $n = 2$ case with $v_{11} = 600, v_{12} = 400, v_{21} = 400$, and $v_{22} = 600$. Then, by assigning room 1 to agent 1 and room 2 to agent 2, any $p = (p_1, p_2)$ with $400 \leq p_1 \leq 600$ and $p_2 = 1000 - p_1$ gives an envy-free solution. However, most would agree in this case that the "fair" solution would be $p_1 = p_2 = 500$. In general, we seek solutions which minimize the following quantity (referred to by [4] as *pettiness*):

[4]This statement requires applying the stronger version of the Miserly Tenants condition as described in Section 8 of [3].

$$\max_{i,j\in I}(v_{i\sigma(i)} - p_{\sigma(i)}) - (v_{j\sigma(j)} - p_{\sigma(j)}) \tag{3}$$

Our goal becomes to find an algorithm which minimizes pettiness subject to envy-freeness. The algorithm should also yield nonnegative prices if possible. Lemma 2 gives us a method to do just this.

Theorem 3. *Minimum pettiness across all envy-free solutions can be achieved from any socially optimal allocation. Furthermore, the same holds when restricting to nonnegative envy-free solutions, if any exist.*

Proof. Suppose the minimum pettiness D is achieved by the solution (σ, p), and let σ' be socially optimal. By Lemma 2, we can fix a permutation p' of p such that $v_{i\sigma(i)} - p_{\sigma(i)} = v_{i\sigma'(i)} - p'_{\sigma'(i)}$ for every i. Then $\max_{i,j\in I}(v_{i\sigma(i)} - p_{\sigma(i)}) - (v_{j\sigma(j)} - p_{\sigma(j)}) = \max_{i,j\in I}(v_{i\sigma'(i)} - p'_{\sigma(i)}) - (v_{j\sigma'(j)} - p'_{\sigma'(j)}) = D$, so the first part of the theorem follows. The second part of the theorem holds by the same argument, since permutations of a nonnegative vector remain nonnegative. □

This leads us to the following polynomial-time algorithm: first, find a socially optimal allocation σ using, say, the $O(n^3)$ Hungarian algorithm [5]. Then solve the following linear program to obtain envy-free prices minimizing pettiness (we first try the program with the nonnegative constraints, and then remove them if needed.) The first constraint guarantees that pettiness is bounded by the variable being minimized, y. The second constraint ensures envy-freeness, and the third ensures feasibility.

$$\begin{array}{lll}
\text{minimize} & y & \\
\text{subject to} & |(v_{i\sigma(i)} - p_{\sigma(i)}) - (v_{j\sigma(j)} - p_{\sigma(j)})| \le y & \forall i,j \in I \\
& v_{i\sigma(i)} - p_{\sigma(i)} \ge v_{i\sigma(j)} - p_{\sigma(j)} & \forall i,j \in I \\
& \sum_{k\in R} p_k = P & \\
& p_k \ge 0 & \forall k \in R
\end{array}$$

3.2 Dividing Goods

Spliddit's second application allocates a set of *indivisible* goods G to a set of n players; a solution is a list $(A_1, \dots, A_n)$ where the A_i's partition G. Inheritance is the paradigmatic use case, e.g., dividing an art or jewelry collection among three or more heirs. Each player i has value v_{ij} for good j. As mentioned in Section 2, we assume that valuations are *additive*, that is, the value of player i for a bundle of goods X is $v_i(X) \triangleq \sum_{j \in X} v_{ij}$. So, each user i reports v_{ij} for $j \in G$.

In the divisible goods case, the two competing solutions are the Competitive Equilibrium from Equal Incomes (CEEI) and the Egalitarian Equivalent (EE) [6]. Both solutions are Pareto efficient; the CEEI solution is also *envy-free*, while the EE solution is *equitable.* Let us denote an allocation of the goods by $A_1, \dots, A_n$, where A_i is the bundle of goods allocated to player i. In this setting, a solution is envy-free if for every pair of players $i, j \in I$, $v_i(A_i) \geq v_i(A_j)$. A solution is equitable if for every $i, j \in I$, $v_i(A_i) = v_j(A_j)$.

While the EE and CEEI solutions are guaranteed to exist in the divisible case, envy-freeness and equitability are not always feasible in the indivisible case; this is especially obvious when $|G| < n$. However, Moulin shows that when $n = 2$, both solutions involve splitting at most one good [6]. Because we believe it is always reasonable to split a single good (which can be achieved by either time sharing or selling the good and sharing the profits), Spliddit's Dividing Goods app uses the EE solution for $n = 2$. EE was chosen over CEEI because when $n = 2$, equitability implies envy-freeness. For an efficient algorithm to compute EE solutions when $n = 2$, see [7].

Unfortunately, there is no bound on the number of goods split by EE and CEEI for $n > 2$, so we turn elsewhere to handle this case. Again, fix an allocation $A_1, \dots, A_n$. Spliddit's algorithm considers three increasingly weaker levels of fairness:

1. *Envy-freeness:* (Defined above.) It is clear that envy-freeness is not always feasible.

2. *Proportionality:* $v_i(A_i) \geq v_i(G)/n$. In Spliddit, this means each player assigns at least $1000/n$ points to her bundle. Again, clearly proportionality may not be feasible.

3. *Maximin share guarantee:* The *maximin share (MMS) guarantee* of player i is

$$\text{MMS}(i) = \max_{X_1,\ldots,X_n} \min_j v_i(X_j),$$

   where the max is taken over partitions of the items into n subsets $X_1, \ldots, X_n$. Intuitively, this is the value player i could guarantee if she divided the items into n bundles, but then selected a bundle last. An MMS allocation satisfies $v_i(A_i) \geq \text{MMS}(i)$ for all players i. Although an MMS allocation may not exist, counterexamples are elaborate and extremely unlikely to occur in practice. Moreover, an approximate MMS allocation is guaranteed to exist. Specifically, we can always find an allocation such that $v_i(A_i) \geq \frac{2}{3}\text{MMS}(i)$ [8].

The notions are increasingly weak in the sense that envy-freeness implies proportionality, and proportionality implies maximin share guarantee. Spliddit's algorithm works as follows. First, it finds the highest feasible level of fairness. If envy-freeness and proportionality are infeasible, the algorithm computes the maximum $\alpha > 0$ such that each player can achieve an α fraction of her MMS guarantee. Second, the algorithm maximizes social welfare — $\sum_i v_i(A_i)$ — subject to the fairness constraint found in the first phase. Crucially, while we believe that it will always be possible to find an MMS allocation (with $\alpha = 1$), $\alpha \geq 2/3$ is provably feasible even in the worst case [8]. This fact enables us to specify an indisputable fairness guarantee, honoring Spliddit's promise to provide *provably fair solutions.*

3.3 Assigning Credit

The third application divides credit for a joint project between n players. Possible use cases include determining scientific credit for a paper, dividing a company bonus based on employees' relative contributions, or sharing credit for a class project. Player i reports how *the rest* of the credit should be distributed among the other players. For example, if player 1 thinks that players 2 and 3 contributed equally, and player 4 contributed twice as much, then player 1 would report the contributions 25%, 25%, and 50%, respectively.

A solution divides 100% of the full credit for the project among the players.

Work by de Clippel et al. provides a family of rules for credit division; Spliddit implements one of them [9].[5] This solution is guaranteed to satisfy a slew of desirable properties. Most importantly, it is *impartial*: a player's share of the credit is independent of her report. The solution is also *consensual*: if there is a division that agrees with all players' reports, then it is the outcome. While consensuality seems to be quite unrestrictive at first glance, de Clippel et al. show that if $n = 3$, an impartial and consensual rule cannot be *exact*, that is, it would have to sometimes allocate less than 100% credit overall.[6] Spliddit therefore enforces $n \geq 4$.

It is worth noting that one of the possible use cases of the credit division application — sharing scientific credit for a paper — gives rise to interesting questions. Determining the order of authors is a notorious source of acrimony in scientific fields where contribution-based ordering is the norm (that is, almost all scientific fields). Ordering authors by their (fair) share of the credit is an obvious solution. But doing so would not preserve impartiality! Intuitively, while a player cannot change her own share of the credit, she can decrease another player's share below her own, thereby increasing her position in the author ordering. Very recent work by Berga and Gjorgjiev establishes impossibility results for impartial ranking under a strong notion of impartiality [10]. Despite this theoretical difficulty, we do believe it is reasonable (albeit not ideal) to use Spliddit's credit division application for the explicit purpose of ordering authors.

3.4 Splitting Fare

The fourth application involves splitting the cost of a taxi or other ridesharing service (for example, Uber or Lyft) between multiple passengers who need to be dropped off at different locations. Let $I = \{1, 2, \ldots, n\}$ be our set of agents (passengers), and suppose we have a function $f : (I \cup \{\text{pickup}\}) \times I \Rightarrow \mathbb{R}_+$ which computes the fare between every

[5]Specifically, the formula in Equation (17) of the paper of de Clippel et al. is used, with arithmetic means as the aggregators ρ and τ.

[6]Note that normalizing the shares would violate impartiality: by changing the sum of shares, a player can change her own normalized share of the credit.

pair of passengers, as well as the fare from the initial pickup location to each passenger[7]. For each $i \in I$, $f(i,i) = 0$. Then, a solution is a pair (σ, p) where σ is a permutation of I (interpreted as a route), and $p = (p_i)_{i \in I}$ records how much each passenger contributes to the total fare. In any valid solution, the total amount paid must equal the fare of the entire route[8]:

$$\sum_i p_i = f(pickup, \sigma(1)) + \sum_{i=1}^{n-1} f(\sigma(i), \sigma(i+1)) \tag{4}$$

At first glance, it seems that simply dividing the total fare in proportion to $f(\text{pickup}, \sigma(i))$ for each i is a good approach. However, as illustrated in Figure 7, there are scenarios in which the proportional method yields unintuitive and arguably unfair outcomes.

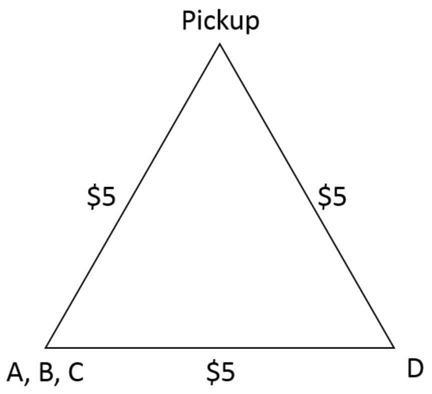


Figure 7: By the proportional method, each agent would be asked to split the total fare of \$10 evenly: \$2.50 each. However, since D lives far away from everyone else, D has a large impact on the total fare. In contrast, if any one of A, B, and C wasn't in the taxi, the cost of the cheapest route would stay the same.

Instead, we turn to the Shapley value, described by Moulin as *the most important contribution of game theory to distributive justice* [6]. To apply the Shapley value, we view the problem as a cooperative game with players I. For each $S \subset I$, the cost $C(S)$ of the coalition S is the minimum fare required to drop off everyone in S. In symbols,

[7]Spliddit uses the TaxiFareFinder API, a service which estimates the cost of taxis and rideshares such as Uber and Lyft worldwide, to compute f. To reduce burden on their servers, we assume f is symmetric.

[8]In reality, this formula is a bit simplistic, as taxis usually charge a one-time base fare known as the "flag drop" charge. Spliddit takes this into account, but for simplicity it is omitted from the presentation.

$$C(S) = \min_{\sigma}(f(pickup, \sigma(1)) + \sum_{i=1}^{|S|-1} f(\sigma(i), \sigma(i+1))) \tag{5}$$

where the minimization is over all permutations σ of S. Then, the Shapley value assigned to agent $i \in I$ is defined as follows:

$$s_i = \frac{1}{n!}\sum_{\sigma}(C(P_i^{\sigma} \cup \{i\}) - C(P_i^{\sigma})) \tag{6}$$

where the summation is over all permutations σ of I and P_i^{σ} is the set of agents preceding i in σ. The Shapley value of agent i can be interpreted as her expected marginal cost over all ordering of agents. In the example from Figure 7, A, B and C each have Shapley value \$1.67 while D has a value of \$5.00. The Shapley value is characterized as the unique value satisfying the following two properties for all cooperative games [6]:

1. *Symmetry*: if agents i and j are equivalent in the sense that for every $S \subset I \setminus \{i, j\}$ $C(S \cup \{i\}) = C(S \cup \{j\})$, then $s_i = s_j$.

2. *Marginalism*: s_i is dependent only on i's marginal contributions to the cost of every subset $S \subset I \setminus \{i\}$

There are two more appealing properties of the Shapley value when applied to the Splitting Fare application. The first, as shown in Lemma 4, is that whenever C is *subadditive* ($\forall S, T \subset I, S \cap T = \emptyset \Rightarrow C(S \cup T) \leq C(S) + C(T)$), the value satisfies *individual rationality*: $s_i \leq f(\text{pickup}, i)$. That is, no passenger pays more than what she would pay if she took a taxi alone. In practice, C will typically be subadditive since subadditivity of costs is the main reason to share rides in the first place!

Lemma 4. *If C is subadditive, $s_i \leq f(pickup, i)$ for all i.*

Proof.

$$\begin{aligned}
s_i &= \frac{1}{n!}\sum_{\sigma}(C(P_i^\sigma \cup \{i\}) - C(P_i^\sigma)) \\
&\le \frac{1}{n!}\sum_{\sigma}(C(P_i^\sigma) + C(\{i\}) - C(P_i^\sigma)) && \text{[By subadditivity]} \\
&= \frac{1}{n!}\sum_{\sigma} C(\{i\}) \\
&= \frac{1}{n!}\sum_{\sigma} f(\text{pickup}, i) && \text{[By definition of C in Eq. (5)]} \\
&= f(\text{pickup}, i) && \text{[There are } n! \text{ permutations of } I\text{]}
\end{aligned}$$

□

One final noteworthy property of the Shapley value for our setting is *linearity*: if we describe another cooperative game where $C'(S) = \alpha C(S)$ for all $S \subset I$, then the new Shapley value for agent i is αs_i. This is important for Spliddit because the function f is only an estimate, and may be inaccurate due traffic conditions, weather conditions, or discrepencies in cost per mile among different taxi companies. As long as these inaccuracies result in a roughly linear scaling of the actual cost function, linearity implies that all passengers are affected roughly equally.

Despite being conceptualized in 1953 [11] and appearing in thousands of research papers since, the Shapley value has yet to be implemented in a publicly available service to the author's knowledge. In general, the Shapley value is difficult to apply because it requires computing the cost of every subset of agents. However, for the Splitting Fare application each of these costs are well defined, and as long as n is reasonable small can be computed quickly. We hope that this application brings additional welcomed attention to the value.

3.5 Distributing Tasks

The fifth and final application allocates a set of undesirables (tasks, work shifts, chores, etc.) to a set of n players. Let $I = \{1, 2, \ldots, n\}$ be our set of agents, T be a set of tasks, and associate with each $t \in T$ a quantity $q_t \in \mathbb{N}_+$ corresponding to the number of times t is to be completed. A solution is a nonnegative matrix $Q = (q_{it})_{i \in I, t \in T}$ such that $\sum_i q_{it} = q_t$ for each task t. Here, q_{it} is interpreted as the number of times agent i is to complete task t. Unlike the Dividing Goods application, it is likely the the Distributing Tasks application will be used for tasks which are repeated frequently, such as on a daily, weekly, or monthly basis. For this reason, we believe that it's reasonable to treat tasks as divisible and then convert fractional assignments into pure assignments via randomization. We'll use the EE solution for distributing tasks, partially due to relative ease of computation compared to CEEI.

For each player i and task t, v_{it} represents the fraction of the total work i assigns to a single unit of task t.[9] In particular, $\sum_{t \in T} q_t v_{it} = 1$ for each i. Valuations are assumed to be additive: that is, q units of task t and q' units of task t' constitute a $qv_{it} + q'v_{it'}$ fraction of the total work according to i.

The EE solution $Q^* = (q^*_{it})_{i \in I, t \in T}$ satisfies $\sum_t q^*_{it} v_{it} = \sum_t q^*_{jt} v_{jt}$ for each i, j as well as Pareto efficiency. Note that in the tasks setting, an allocation is Pareto efficient if there is no other allocation in which one agent believes she receives strictly less work, while all other agents believe that they receive weakly less work. We'll show that the following linear program arrives at the EE solution.

$$\begin{array}{lll} \text{minimize} & \sum_{i \in I, t \in T} q_{it} v_{it} & \\ \text{subject to} & \sum_{i \in I} q_{it} = q_t & \forall t \in T \\ & \sum_{t \in T} q_{it} v_{it} = \sum_{t \in T} q_{jt} v_{jt} & \forall i, j \in I \\ & q_{it} \geq 0 & \forall i \in I, t \in T \end{array}$$

First, observe that there is always a feasible solution: $q_{it} = \frac{q_t}{n}$ works. Second, observe that by the second constraint, any solution to this program will be equitable. So, the following theorem completes the proof that the linear program computes the EE solution.

Theorem 5. *The linear program yields a Pareto optimal solution.*

Proof. Since the EE solution is guaranteed to exist [8], there is some equitable solution which is also pareto optimal. Fix such a solution Q, and let u be the utility of each

[9]On the website, agent i reports the ratios $\frac{v_{it}}{v_{it'}}$ which can be interpretted as the number of units of task t agent i views as equivalent to a single unit of task t'.

agent under Q. Let Q_{LP} be the solution output by the linear program, and u_{LP} the utility assigned to each agent under Q_{LP}. Since Q is Pareto efficient, $u \leq u_{LP}$. However, $u < u_{LP}$ is impossible, since then the linear program would then be able to achieve a smaller objective value of $nu < nu_{LP}$. We conclude $u = u_{LP}$. Since Q_{LP} yields identical utilities as a Pareto efficient allocation, Q_{LP} is Pareto efficient as well. $\square$

The output of the Distributing Tasks application must be an assignment of whole tasks to agents, but each q_{it} may be fractional in the EE solution. The approach we take is to view each q_{it} as the expected number of times agent i is assigned task t, and then round each q_{it} up or down via a lottery. With this approach, all of our fairness guarantees are in expectation; however, the actual allocation will differ from the expected allocation by at most one of each task. Budish et al. proves that such a lottery exists[10], and provides an algorithm which implements the lottery in polynomial time [12].

4 Implementation Details

The implementation of Spliddit is centered around a single web application written in Ruby on Rails. All of Spliddit's *nouns*, including application instances, players, goods, valuations, and allocations, are modeled using Ruby on Rails Active Record, an object-relational mapping system. On the front end, Spliddit uses the jQuery Javscript library to create interactive user interfaces such as the one used for bidding. Spliddit takes advantage of several services offered by Amazon's cloud computing platform: Elastic Compute Cloud (for hosting Spliddit's webservers and running the division algorithms), Elastic Beanstalk (for deploying and automatically scaling Spliddit to handle heavy web traffic), Relational Database Service (for hosting Spliddit's database), and Simple Email Service (for sending email notifications). Figure 8 provides a diagram of Spliddit's server architecture.

[10]Budish [12] shows that an expected assignment can be implemented by a distribution over pure assignments as long as the constraints form a so-called *bihierarchy*. In our case, we can use one constraint hierarchy to make sure that each task is fully assigned, and another to ensure that each agent i receives an amount of task t within one unit of q_{it}.

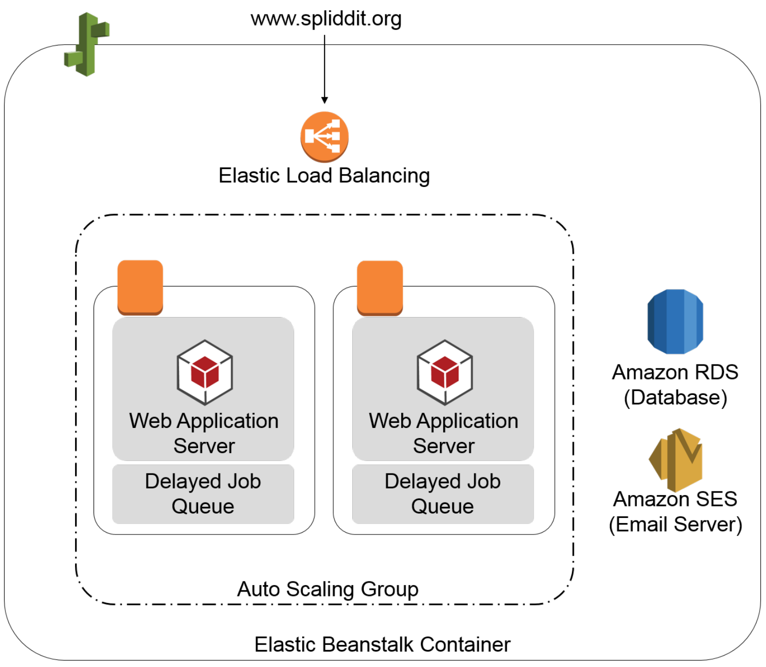


Figure 8: A diagram of Spliddit's server architecture, which is hosted on Amazon's cloud computing platform.

The algorithms described in the previous section are written in Java for improved performance and ease of implementation. All linear and integer linear programs are modeled and solved using IBM's CPLEX Optimizer and IBM's Concert Technology. For the $n \geq 3$ case of the Dividing Goods application, each fairness constraint is attempted in turn from strongest to weakest; however, for the weakest level of fairness, the algorithm must first solve integer linear programs to compute each player's MMS guarantee. In order to free the webservers to quickly respond to incoming HTTP requests, Spliddit uses Delayed Job to run the algorithms in background processes.

5 Reception

Spliddit officially launched on November 4, 2014. Following launch, Spliddit received press coverage in a variety of popular technology and science websites such as Gizmodo, Lifehacker, Slashdot, and Fast Company. Building on this coverage, Spliddit received nearly 40,000 unique visitors, who combined to use the three launch application (Sharing Rent, Dividing Goods, and Assigning Credit) over 9,000 times. Feedback has been overwhelmingly positive, with many users commending the interfaces, ease-of-use, and overall mission of Spliddit. Some of our favorite feedback has come from teachers and professors in quantitative fields who commented on how Spliddit is able to demonstrate the beauty and impact of mathematical research. To this end, Spliddit has also been mentioned on social media by the Mathematical Association of America and the National Science Foundation.

Through user-submitted comments and social media posts, we've also been able to get a better grasp on how people are using our applications. The Assigning Credit application has been applied in many interesting ways, including splitting bonuses at both small and very large companies, sharing profits from craft projects, ordering authors on scientific papers, and monitoring group dynamics in school projects at both the high school and university level. Out of the three applications, the Sharing Rent has been the most popular, having been used over 5,000 times.

Finally, many users submitted ideas new ideas for new applications; the most popular two suggestions evolved into the Splitting Fare and Distributing Tasks applications, which launched on April 28, 2015. In particular, the Distributing Tasks application arose due to several requests for dividing different kinds of shifts (evening, weekend, holiday) in hospital settings. Spliddit will continue to evolve based on feedback from users.

6 Closing Remarks: Empirical Fair Division Research

As discussed above, Spliddit's two primary goals are providing access to fair division methods, and outreach. However, Spliddit also has the potential to become a revolutionary platform for empirical fair division research. Indeed, as noted by Herreiner and Puppe [13], fairness properties such as envy-freeness are difficult to study in the lab. A typical experiment in the context of indivisible goods informs each participant of her "value" for each virtual "good", and pays each participant based on her value for her allocated bundle. In this setting envy-freeness is problematic, because a participant does not truly care about which goods were allocated to another participant (as the goods have no real value) — presumably she mainly cares about how much other participants were paid.

In contrast, Spliddit allows us to partition thousands of users (who report their values for real goods) into multiple groups, and employ a different solution for each group. Happiness surveys will then allow us to gage the relative importance of various criteria in a way that was previously impossible.

7 References

1. Klijn, Flip. ”An algorithm for envy-free allocations in an economy with indivisible objects and money.” Social Choice and Welfare 17.2 (2000): 201-215.

2. Brams, Steven J., and D. Marc Kilgour. ”Competitive fair division.” Journal of Political Economy 109.2 (2001): 418-443.

3. Su, Francis Edward. ”Rental harmony: Sperner's lemma in fair division.” American Mathematical Monthly (1999): 930-942.

4. Gal, Ya'akov, Mash, Moshe, Procaccia, Ariel D., and Yair Zick. “Equitability and Envy-Freeness Tradeoffs in Rent Division.” Manuscript, 2015.

5. Wikipedia: http://en.wikipedia.org/wiki/Hall's_marriage_theorem

6. Moulin, Hervé. Fair division and collective welfare. MIT press, 2004.

7. Brams, Steven J., and D. Alan Taylor. “Adjusted Winner Website.” http://www.nyu.edu/projects/adjustedwinner/.

8. Procaccia, Ariel D., and Junxing Wang. ”Fair enough: Guaranteeing approximate maximin shares.” Proceedings of the fifteenth ACM conference on Economics and computation. ACM, 2014.

9. De Clippel, Geoffroy, Herve Moulin, and Nicolaus Tideman. ”Impartial division of a dollar.” Journal of Economic Theory 139.1 (2008): 176-191.

10. Berga, D. and Riste Gjorgjiev. “Impartial Social Rankings.” Manuscript, 2014.

11. Shapley, Lloyd S. ”Stochastic games.” Proceedings of the National Academy of Sciences of the United States of America 39.10 (1953): 1095.

12. Budish, Eric, et al. ”Designing random allocation mechanisms: Theory and applications.” The American Economic Review 103.2 (2013): 585-623.

13. Herreiner, D. K. and Puppe, C. D. “Envy freeness in experimental fair division problems.” Theory and decision 67.1 (2009), 65-100.