Xi'an Jiaotong-Liverpool University

西交利物浦大学

DTS311TC Final Year Project

Investigating Fair Allocation Algorithms to Put them into Wider Use through an Easy-to-Use Interface

In Partial Fulfillment of
the Requirements for the Degree of
Bachelor of Engineering

By

Yongteng Lei

ID: 1930236

Supervisor Name

Dr. Md Maruf Hasan

School of AI and Advanced Computing

XI'AN JIAOTONG-LIVERPOOL UNIVERSITY

April 2023

Abstract

This FYP will be devoted to investigating various algorithms for allocating a number of indivisible items that two agents have different preferences on. In fact, the project treats the problem of fair allocation as a social choice problem, focusing more on real-life scenarios than on game theory. In other words, the project treats preference information as the real thoughts of two agents in a given scenario, regardless of individual reasons for hiding or misrepresenting facts. Those algorithms are then applied to an easy to use interface, in this case a web page, as a reference for solving problems in real everyday situations, e.g., inheritance of items, distribution of surplus items, task allocation, etc.

Keywords: Fair allocation algorithms; Algorithm implementation; Web development; User interface; Golang;

Contents

List of Figures

List of Tables

Glossary

A

AW: *Adjusted Winner Algorithm*.

D

DC: *Divide and Choose Algorithm*.

E

EF: *Envy-freeness*. EF ensures no participant envies another's allocation.

EF1: *Envy-freeness up to one*. When an agent envies someone else, but the envy can be eliminated by removing one of other's (most valuable) goods.

$EF1_2$: *Envy-freeness up to one Algorithm*.

EFx: *Envy-freeness up to any items*. When an agent envies someone else, but the envy can be eliminated by removing any of other's goods.

M

MMS: *Maximin Share Guarantee*.

MNW: *Maximum-Nash-Welfare*.

N

NP problem: *Non-deterministic Polynomial Problem*. A computational task that can be verified in polynomial time by a deterministic algorithm.

NP-hard problem: *Non-deterministic Polynomial-hard Problem*. A problem that is at least as hard as any NP problem, meaning any NP problem can be reduced to it in polynomial time. NP-hard problems are considered the toughest in NP.

R

RR: *Round Robin Algorithm*.

Chapter 1

Introduction

Fairness is what people have been pursuing in theory and practice [1]. The desire to create allocations that satisfy both efficiency and fairness criteria based on the preferences of the participants involved has driven the development of algorithms in this area. With the change of time, fair division theory has been studied in mathematics, economics, political science, computer science and other fields with different perspectives and purposes. It is not surprising that an fair allocation of resources would have a vital position in collective choice settings.

Allocating infinitely divisible resources fairly is difficult enough, but things become significantly more difficult when the resources consist of a finite collection of indivisible goods [2]. Each agent always receives a subset of resources referred to as "packs", along with their evaluation of these items. An allocation must be made based on the agent's preferences for these packs, taking into account one or more criteria for fairness and efficiency. Consider the "Two for three candy" [1] case, if there is no reasonable fairness criterion, and no compromise by an agent, the distribution will always be a tie (there is always an agent with more candy). To demonstrate how challenging a fair item distribution could be, we simplify the scenario by assuming that the agents' utility are linear, with no synergies[2], and consider the "Santa Claus problem", how to distribute a finite number of gifts to a finite number of children, while maximizing the utility of no unhappy children? This problem is an $NP\text{-}hard$ problem, even if only one fairness criterion is considered. And it is believed that there is no approximate efficient solution [3].

The distribution of items that cannot be divided into smaller parts is an important research topic that cannot be overlooked. This is partly because there may not exist a "fair"

[1] Two agents pursue fairness in the allocation of three indivisible items

[2] In order to simplify the problem of allocating items, it is common to assume that all items are independent (so they are neither substitutes nor complements).

distribution in such scenarios and partly because some straightforward algorithms fail due to the indivisible nature of the items, which is a common occurrence in real-life situations. A simple way to do this is to line up the agents, cyclically giving them the items they currently want most in order of preference. This sounds very reasonable and convincing, but how to queue them up can be a problem. Fortunately, many talented authors have recently proposed a variety of problem-specific allocation algorithms that satisfy certain fairness and efficiency criteria.

This FYP will investigate the algorithms and concepts of fair allocation, while considering the possibilities of implementation, and transforming them into an easy-to-use interface that can be used as a reference for everyday allocation problems. [4] [5] [6] [7] [8].

Chapter 2

Preliminaries

The FYP will focus on the setting of fair allocation of **indivisible** goods. Consider a fair allocation problem entity $E = (v, M, N)$. M is the set of goods m, which may contain only indivisible goods. N is the set of n agents. $v = (v_1, v_2, v_3, \cdots, v_n)$ is a non-negative *additive valuations* vector, with agent $i \in N$ for set $S \subseteq M$ being $v_i(S) = \sum_{j \in S} v_i(j)$, where $v_i(j)$ denotes the set of agents i for valuation of good $j \in M$, S are partitions of the goods into n packs. The work is to try to assign $|M|$ goods to $|N|$ agents fairly in this entity E according to the corresponding fairness criteria, popularly known as ***MMS***, ***PFS*** and *envy-free* etc.

Definition 1 (Maximin share guarantee)**.** The MMS guarantee of agent $i \in N$ is

$$MMS(i) = \max_{S_1, \cdots, S_n} \min_{j \in N} v_i(S_j)$$

This means that the agent will divide the goods into n packs, and he will get the most desired but the least bundle. It is worth noting that maximin share allocation (which guarantees that every agent gets a share of his / her MMS) may not always exist. Fortunately, however, Procaccia and Wang [9] proved that at least ($\frac{2}{3}$)-approximation maximin share allocation (each agent will get $\frac{2}{3}$ of his/her MMS) is always available.

Definition 2 (Proportional fair-share)**.** The PFS guarantee of agent $i \in N$ is $\frac{1}{n}$ of his or she utility from the entire goods set M, namely $PFS = \frac{v_i(M)}{|N|}$. A proportional allocation is that for each agent in N, have:

$$v_i(S_i) \geq \frac{v_i(M)}{n}$$

Each agent receives a PFS of (total value / n). It is worth noting that *PFS allocation* always implies MMS if every agent with superadditive utility[10].

Definition 3 (Envy-freeness)**.** EF indicates that all agents weakly prefer their own bundle over others, have

$$v_i(S_i) \geq \alpha * v_i(S_j)$$

for two agent i and j, where $\alpha \in (0, 1]$

When $\alpha = 1$, so called *EF*.

Definition 4 (Envy-freeness-except-1)**.** When agent i envies j, but the envy is eliminated by removing one of j's (*most* valuable) goods, then EF1, have

$$v_i(S_i) \geq \alpha * v_i(S_j \setminus g)$$

for two agent i and j, where $\alpha \in (0, 1]$, and the "\" symbol represents remove a good g from packs S_j

When $\alpha = 1$, so called *EF1*. EF1 is weaker than EF, and the Maximum Nash Welfare algorithm is both EF1 and Pareto-efficient [11].

Definition 5 (envy-free up to at most any item)**.** When agent i envies j, but the envy is eliminated by removing one of j's (*least* valuable) goods, then EFx, have

$$v_i(S_i) \geq \alpha * v_i(S_j n g)$$

for two agent i and j, where $\alpha \in (0, 1]$

When $\alpha = 1$, the allocation criterion is referred to as *EFx*. Although EFx is weaker than EF, it is strictly more demanding than EF1. In terms of fairness, EFx can be considered more reasonable than EF1; however, it is also harder to achieve. The general availability of EFx allocation remains an open question, and the minimum number of agents required in cases with additive valuations is four [12].

The allocation of an entity E as $v = (\emptyset, \emptyset, \cdots, \emptyset)$ still satisfies EF, EFx, and EF1. However, making all items M a charity does not make any sense to the agent, although no one would be envy of a charity set [13]. Obviously, we also need some efficiency metrics built on top of pure envy-free.

Definition 6 (Pareto efficient)**.** Pareto efficiency or Pareto optimality (PO) is the case where there is no alternative allocation v' such that $v_i(S_{i'}) \geq v_i(S_i)$, for all $i \in N$.

In other words, there is no PO allocation in which an agent $i \in N$ can improve his or her value without affecting the value of others.

These preliminaries will always be in mind, and will serve as the basis for exploring and implementing fair allocation algorithms.

Chapter 3

Literature Review

As mentioned earlier, the main concern of FYP is the fair allocation of indivisible goods (The goods will be treated as a whole and cannot be divided into multiple parts), in addition, some websites that are available for fair allocation are also mentioned in this section.

3.1 Indivisible Goods Fair Allocation

When things turn out to be indivisible, there is no guarantee of exact envy-free and fair notions. Therefore, fair relaxations are presented, such as EF1, EFx, MMS, etc., which define "certain" equilibria [11] [14] [15] . There have been too many valid and versatile algorithms proposed by many computer scientists, but here are some of the most commonly used and powerful ones, which are also the basis of algorithms in more complex settings.

3.1.1 Divide-and-Choose (DC)

Divide-and-Choose is one of the most classical assignment algorithms, and in the case of only two agents, the process is intuitive and efficient. The first agent has the power to divide the items into groups, and the second agent will choose, so the most efficient and fair way to divide is for **the first agent to maximize the value of each group as much as possible** to ensure that he or she also reaps the maximum benefit. MMS and always hold for the first agent if it maximizes the smaller group in (X_1, X_2), and EF is valid for the second agent because he or she gets the preferred group from (X_1, X_2). This has been proven by Plaut and Roughgarden [16].

3.1.2 Adjusted-Winner (AW)

Another widely used allocation algorithm for two-agent scenarios is Adjusted-Winner. The algorithm dynamically adjusts the two-agent evaluation of items depending on the obtained utility [17]. EF1 is always present for the first agent to start allocating the item. This algorithm guarantees the highest possible social welfare between the two agents, but does not necessarily fulfill MMS or EFx [18].

3.1.3 Sequential Allocation and Round-Robin (RR)

Another class of fair allocation algorithms that can be implemented efficiently is Sequential Allocation, also known as sequential-picking, where people pick their favorite items in a certain order that remains valid [17]. The Round-Robin algorithm is one of the most popular sequential algorithms, which repeats a sequential pattern 1...n, for indivisible items and chores (but not a mixture of them) to guarantee EF1, but EFx may not always exist. Aziz et al [19]. proposed a double Round-Robin solution for the mixing of indivisible items and chores. Amanatidis et al. [20] and Aziz et al. [21] approximated MMS fairness by a sequential algorithm.

3.1.4 Envy-freeness (EF)

Envy-free (EF) is an idealized fairness criterion that requires the allocation of things to each participant in such a way that they do not envy the allocation of others. Even with a small number of participants and products, achieving this level of fairness is notoriously difficult [4][22]. Early contributions in this field include Stromquist's [23] investigation of the housing allocation problem and Varian's [24] study on fair division in economics.

Researchers have developed more practical fairness criteria, such as EF1 and EFx, in response to the limitations of EF. EF1 requires that each participant's item allocation be such that they do not overly envy another participant's allocation [11]. This requirement is reasonably simple to meet and is deemed enough in many situations. In indivisible item allocation, Plaut and Roughgarden [16] suggested a polynomial-time technique to achieve EF1. Considering the criterion EFx, which is stronger than EF1 but not EF hard to achieve, the participants are required not to envy the allocation of others by removing any of the items, however, the widespread availability of EFx is still an open issue [12].

3.2 Related Works

3.2.1 Spliddit

Caragiannis et al [11]. have shown a clever way convert a divisible problem related algorithm MNWs (Maximum-Nash-Welfare) into an indivisible one, and they proved that MNWs still imply a certain EF1 as well as MMS in the indivisible setting. Finally they found that an MNW is an *APX-hard*, a *NP* optimization problem, in this scenario, and to do so they devised a special trick, giving each agent 1000 points, and using a polynomially reducible function to overcome this difficulty. Lastly, they used this algorithm as the underlying implementation of the Spliddit website (Fig. 3.1).

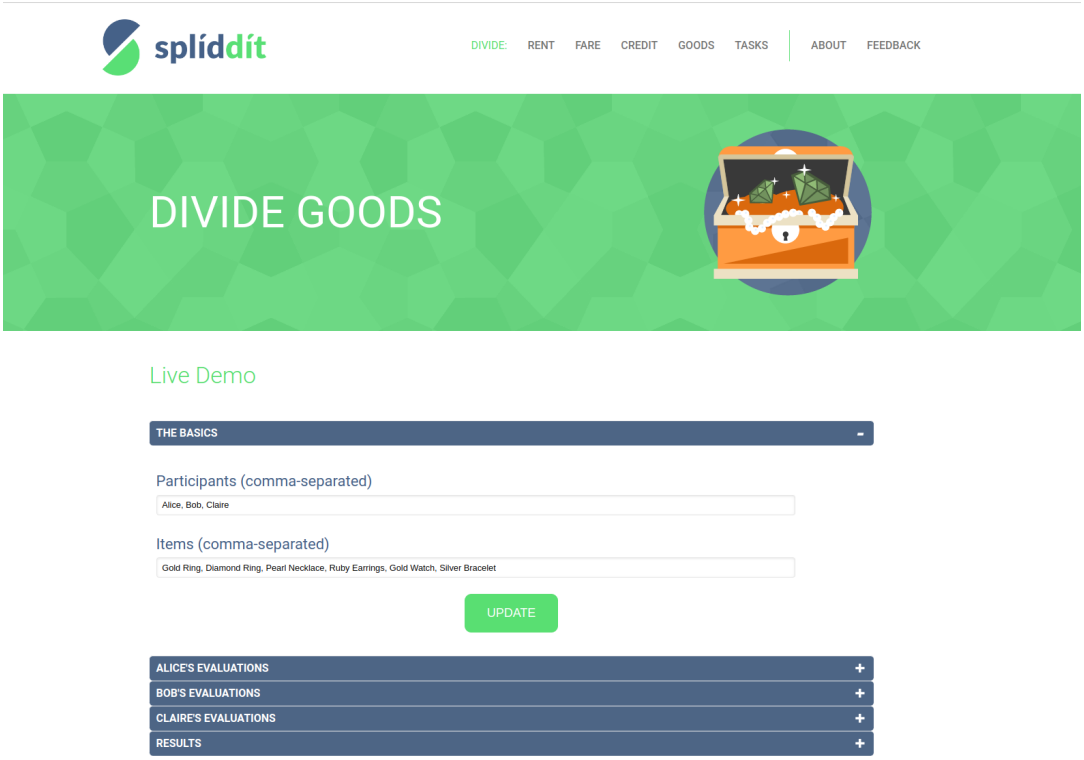
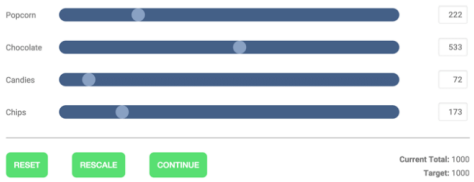
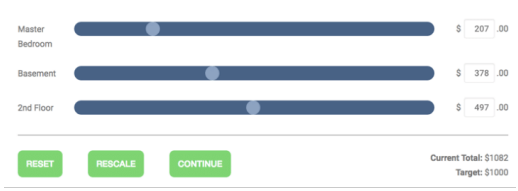


Fig. 3.1: Spliddit-A prominent fair allocation website powered by Carnegie Mellon

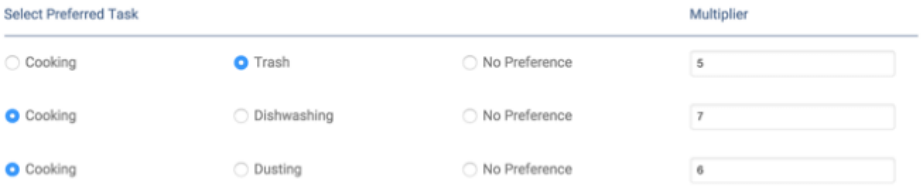
Spliddit was designed to address three specific fair allocation issues: Splitting rent, dividing goods, and sharing credit. Each problem type has detailed explanations and links to educational articles as well as demos. But it is currently out of service.



(a) Spliddit-goods-input-interface



(b) Spliddit-rent-input-interface



(c) Spliddit-tasks-input-interface

Fig. 3.2: Spliddit-input-interfaces

For the goods allocation task, participants are given 1000 points each to indicate their preference for the given goods. The algorithm works on these points, regardless of whether it is based on ordinal utility [1] or cardinal utility [2].

The same thing happens in rent allocation, the difference is that the total number of points is the total amount of rent, and the participants give different offers for different facilities depending on their preferences.

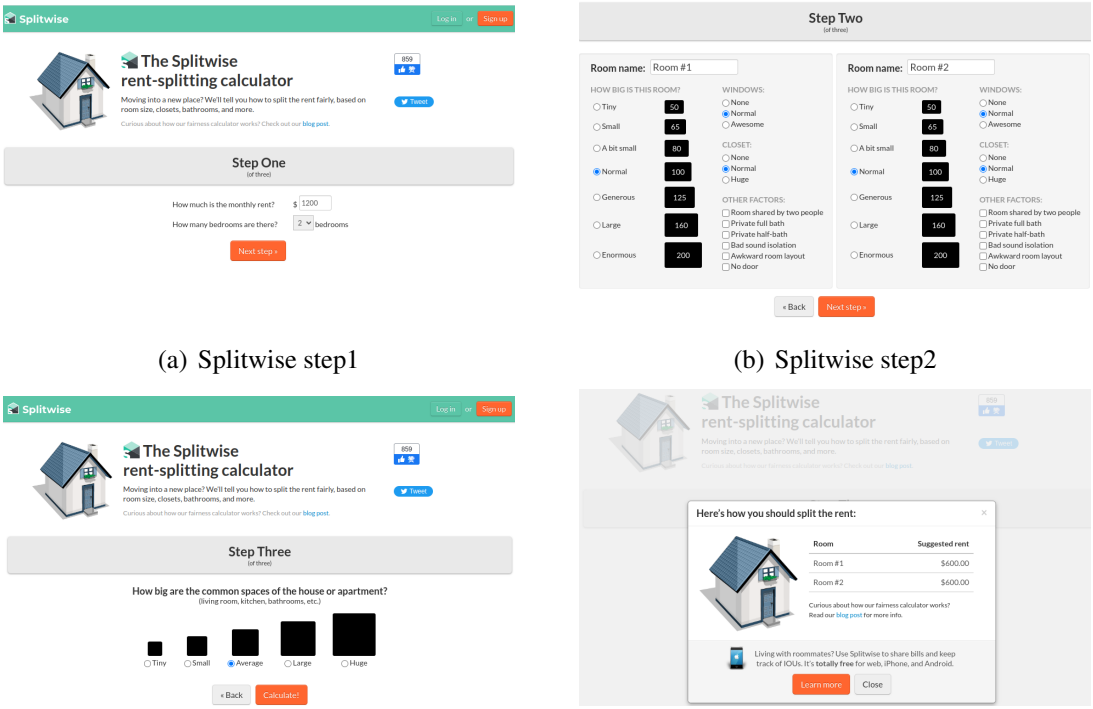
In Spliddit task assignments, multiple users are required to collaborate on a task and each user provides an input signal that is used to compute the result of the task. When multiple users submit input signals, these signals are encoded and selected using a multiplexer, resulting in a combined signal that can be passed to the task algorithm for computation.

3.2.2 Splitwise

The Splitwise Rent Calculator is also a web-based tool that simplifies rent division among roommates, promoting fairness in shared housing situations, which is based on the fairness principles of Nobel Prize-winning economist Alvin Roth, needs users to submit total rent, number of roommates, and room attributes. The rent is then allocated using an iterative process that modifies the rent for each room until all parties agree on the allocation.

[1] Ordinal utility focuses on relative rankings of goods combinations that represent customer preferences, taking just order into account.

[2] Cardinal utility provides numerical values to preferences, offering quantitative comparisons and calculations of satisfaction levels.



(a) Splitwise step1

(b) Splitwise step2

(c) Splitwise step3

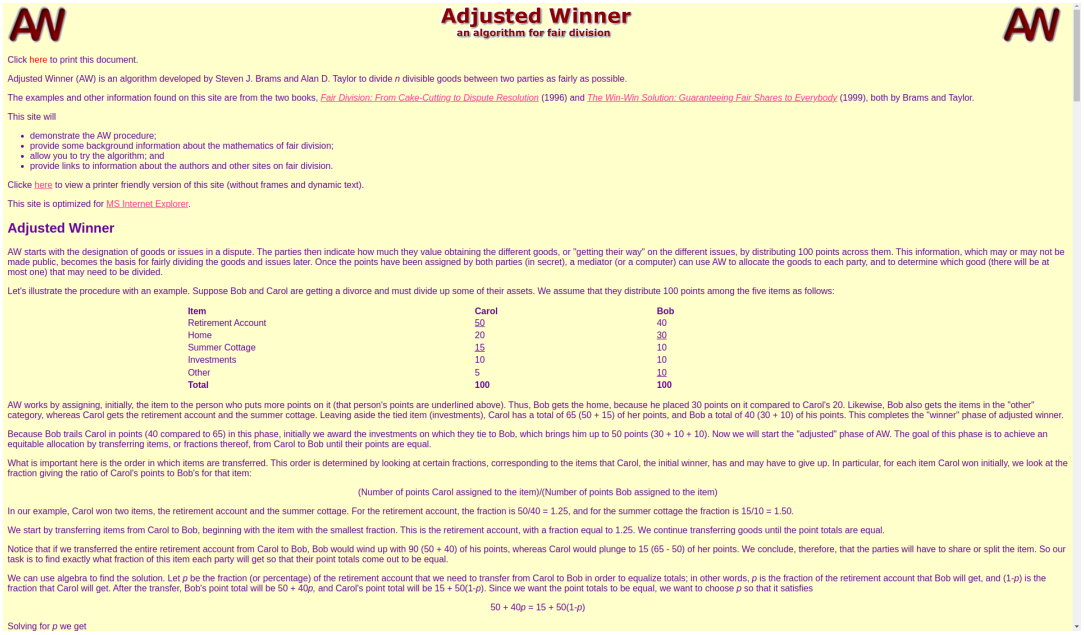
(d) Splitwise result

Fig. 3.3: Splitwise

Spliddit completes the survey of participants' preferences for goods by assigning points, providing great scalability to accommodate both ordinal utility and cardinal utility based algorithms. The simple yet aesthetically attractive interface of the Splitwise Rent Calculator was also an inspiration for web design.

3.2.3 Adjusted Winner Website - NYU

Another website (Fig.3.4), maintained by New York University, demonstrates the process of the Adjusted Winner (AW) algorithm, proposed by Steven J. Brams and Alan D. Dedicated to distribute n divisible items between two participants as fairly as possible [4]. The website also provides some background information on the mathematics of fair allocation, as well as a venue to try out custom data. Although it focuses on divisible items, its pedagogical approach coincides with the expected results of this FYP, providing a reference for fair allocation tasks while offering a general understanding of how the algorithm achieves fairness.



Adjusted Winner
an algorithm for fair division

Click here to print this document.

Adjusted Winner (AW) is an algorithm developed by Steven J. Brams and Alan D. Taylor to divide *n* divisible goods between two parties as fairly as possible.

The examples and other information found on this site are from the two books, *Fair Division: From Cake-Cutting to Dispute Resolution* (1996) and *The Win-Win Solution: Guaranteeing Fair Shares to Everybody* (1999), both by Brams and Taylor.

This site will

- demonstrate the AW procedure;
- provide some background information about the mathematics of fair division;
- allow you to try the algorithm; and
- provide links to information about the authors and other sites on fair division.

Clicke here to view a printer friendly version of this site (without frames and dynamic text).

This site is optimized for MS Internet Explorer.

Adjusted Winner

AW starts with the designation of goods or issues in a dispute. The parties then indicate how much they value obtaining the different goods, or "getting their way" on the different issues, by distributing 100 points across them. This information, which may or may not be made public, becomes the basis for fairly dividing the goods and issues later. Once the points have been assigned by both parties (in secret), a mediator (or a computer) can use AW to allocate the goods to each party, and to determine which good (there will be at most one) that may need to be divided.

Let's illustrate the procedure with an example. Suppose Bob and Carol are getting a divorce and must divide up some of their assets. We assume that they distribute 100 points among the five items as follows:

Item	Carol	Bob
Retirement Account	50	40
Home	20	30
Summer Cottage	15	10
Investments	10	10
Other	5	10
Total	**100**	**100**

AW works by assigning, initially, the item to the person who puts more points on it (that person's points are underlined above). Thus, Bob gets the home, because he placed 30 points on it compared to Carol's 20. Likewise, Bob also gets the items in the "other" category, whereas Carol gets the retirement account and the summer cottage. Leaving aside the tied item (investments), Carol has a total of 65 (50 + 15) of her points, and Bob a total of 40 (30 + 10) of his points. This completes the "winner" phase of adjusted winner.

Because Bob trails Carol in points (40 compared to 65) in this phase, initially we award the investments on which they tie to Bob, which brings him up to 50 points (30 + 10 + 10). Now we will start the "adjusted" phase of AW. The goal of this phase is to achieve an equitable allocation by transferring items, or fractions thereof, from Carol to Bob until their points are equal.

What is important here is the order in which items are transferred. This order is determined by looking at certain fractions, corresponding to the items that Carol, the initial winner, has and may have to give up. In particular, for each item Carol won initially, we look at the fraction giving the ratio of Carol's points to Bob's for that item:

(Number of points Carol assigned to the item)/(Number of points Bob assigned to the item)

In our example, Carol won two items, the retirement account and the summer cottage. For the retirement account, the fraction is 50/40 = 1.25, and for the summer cottage the fraction is 15/10 = 1.50.

We start by transferring items from Carol to Bob, beginning with the item with the smallest fraction. This is the retirement account, with a fraction equal to 1.25. We continue transferring goods until the point totals are equal.

Notice that if we transferred the entire retirement account from Carol to Bob, Bob would wind up with 90 (50 + 40) of his points, whereas Carol would plunge to 15 (65 - 50) of her points. We conclude, therefore, that the parties will have to share or split the item. So our task is to find exactly what fraction of this item each party will get so that their point totals come out to be equal.

We can use algebra to find the solution. Let *p* be the fraction (or percentage) of the retirement account that we need to transfer from Carol to Bob in order to equalize totals; in other words, *p* is the fraction of the retirement account that Bob will get, and (1-*p*) is the fraction that Carol will get. After the transfer, Bob's point total will be 50 + 40*p*, and Carol's point total will be 15 + 50(1-*p*). Since we want the point totals to be equal, we want to choose *p* so that it satisfies

$$50 + 40p = 15 + 50(1-p)$$

Solving for *p* we get

Fig. 3.4: A website showcasing Adjust Winner algorithm maintained by NYU

3.2.4 Fairpy

Fairpy [3] is a Python-based library of fair allocation algorithms, aiming to provide a variety of fair solutions for allocation problems. The library covers several important concepts of fair allocation, including envy-freeness, proportionality, and equitability. It provides a wide range of algorithms for different situations, such as divisible and non-divisible goods, and situations with or without monetary transfers.

Fairpy is designed to be user-friendly, providing clear documentation and examples to facilitate implementation. Researchers, economists, and developers can use the library to explore, analyze, and solve fairness-related problems in a variety of applications, such as resource allocation, scheduling, and collaborative decision-making.

By providing a comprehensive set of tools and resources for Fair division, Fairpy contributes to the ongoing discussion around fairness and equity in both academic and practical contexts.

[3] Find Fairpy at https://github.com/erelsgl/fairpy

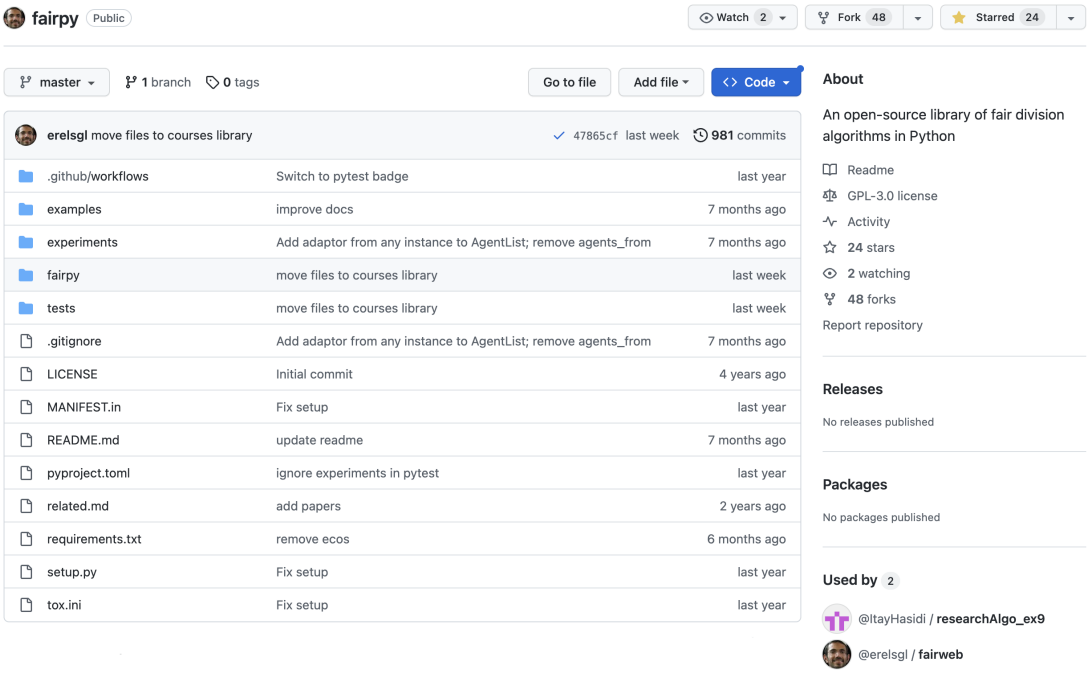


Fig. 3.5: Fairpy – a comprehensive set of tools and resources for fair division

Thanks to such user-friendly interfaces, robust algorithms, and the foundation of economic principles, they are a valuable reference in the field of shared resources and financial management. This FYP will focus on exploring the four widely used ideas and methods mentioned above (DC, AW, RR, EF) as a service with an innovative, easy-to-use, and heuristic website to awaken public awareness of fair allocation and to provide some realistic convenience in life.

Chapter 4

Methodology

4.1 Exploration and Reflection (Theoretical Method)

Fair allocation is a multifaceted topic that spans across various disciplines, including economics, game theory, mathematics, and computer science. Each field tackles unique challenges and approaches solutions in diverse ways to address the issue of different settings.

In order to gain a deep understanding of the problem, a thorough literature review, along with proof and application scenario research, is essential to validate the working process of algorithms. In addition, exploring existing interfaces on the web, such as highly validated and popular blogs and websites, can provide valuable insights for easy-to-understand explanatory content and user-friendly features.

4.2 Design and Implementation (Empirical Method)

This FYP consists of two main parts, namely algorithm exploration and interface implementation. The algorithm section focuses on exploring four widely used algorithms (DC, AW, RR, EF) and trying to implement and test the corresponding allocation behavior using Go language. (There is no pioneer of fair allocation in Go language so far).

For the interface implementation part, there are two accessible ways, namely web-based and terminal applet. The web-based interface aims to create an innovative, eye-catching, easy-to-use, educational fair allocation solution, with strong guidance and robust functionality. The terminal applet is focused on reachability (depending on the features of the terminal, not everyone will be able to use it, but it is always ready to go).

4.2.1 Part1: Algorithm Implementation and Test

Algorithm implementation and innovation

The four selected algorithms are implemented in Go, with DC and RR reproducing the classical implementation. For both AW and EF, their ideas will be taken into account as a basis for implementing two more realistic algorithms, albeit at a kind of compromise of some strict fairness.

Algorithm evaluation criteria

After an in-depth literature review, several algorithms suitable for exploration and implementation at the undergraduate level should be picked and tested for performance in several aspects.

- **Completeness**: Whether the algorithm is able to allocate all goods to all participants.
- **Fairness** [1]: The differences in utility obtained by participants after allocation.
- **Efficiency**: Run time, and number of steps required to complete the allocation.

Algorithm input pattern

In addition, different algorithms may be picky about their inputs. The tests will also consider the performance from 3 different input patterns, to ensure that the algorithm is implemented with some general adaptability.

- **Normal**. All participants have a normal [2] evaluation of the items, and preferences are not approximately the same.
- **Similar**. All participants have approximately the same preference for items.
- **Tie**. All participants have the same preference for items.

Test data preparation

To test the algorithm's performance in equitable allocation cases, it is important to have real-world data, such as the allocation of excess communal goods for the dormitory, divorce asset division, household chores allocation, etc. However, in the absence of such

[1] Different algorithms treat fairness differently, and the differences may even have a large gap. Here it is evaluated by the difference between the utility values obtained between participants after allocation.

[2] Participants independently evaluated the items according to their own preferences, and the evaluations were authentic and free of deception.

data, simulating different scenarios can be a useful alternative to help to evaluate the algorithm's effectiveness. The following test data can be prepared:

- Generate an exhaustive series of N numbers such that the sum is 100 using the program. Choose N values of 4, 5, and 6 (Generating data for 7 evaluations requires more than 32G of memory support). Randomly select 1000 data entries from each series, and test each pair of entries as a set of data for the algorithm. This creates 500 test cases. Simulate the input mode as Normal.
- Repeat the above process but ensure that every 10 data entries have similar preferences. This will create 500 test cases but contain 100 similar patterns as a set of data for the algorithm. Simulate the input pattern as Similar.
- Randomly select 500 data entries and share each data to two participants. This creates 500 test cases, and each participant has an equal claim to goods. Simulate the input pattern as Tie.

Table 4.1: Test Data Settings

Explanation / Input Pattern	**Normal**	**Similar**	**Tie**
# Generated	1000	1000	500
Pair as a case	True	True	False
Similarity	False	Every 10 data	False
# Test cases	500	500	500

By simulating these scenarios, it is possible to test the algorithm's effectiveness in different situations and evaluate its performance.

Overall Analysis

External Testing Analysis with Fairpy After all the algorithms have been implemented and tested, an external test of Fairpy and Fairallol will be carried out to verify two main aspects.

- Whether the same algorithm yields consistent behavior.
- Efficiency comparison (Runtime).

Internal Testing Analysis In addition, a comparative efficiency analysis will be performed within Fairallol using the aspects defined by section 4.2.1 to evaluate the pros and cons of the algorithms implemented and to derive a general summary.

4.2.2 Part2: Interfaces Design and Implementation

Web Interface Design

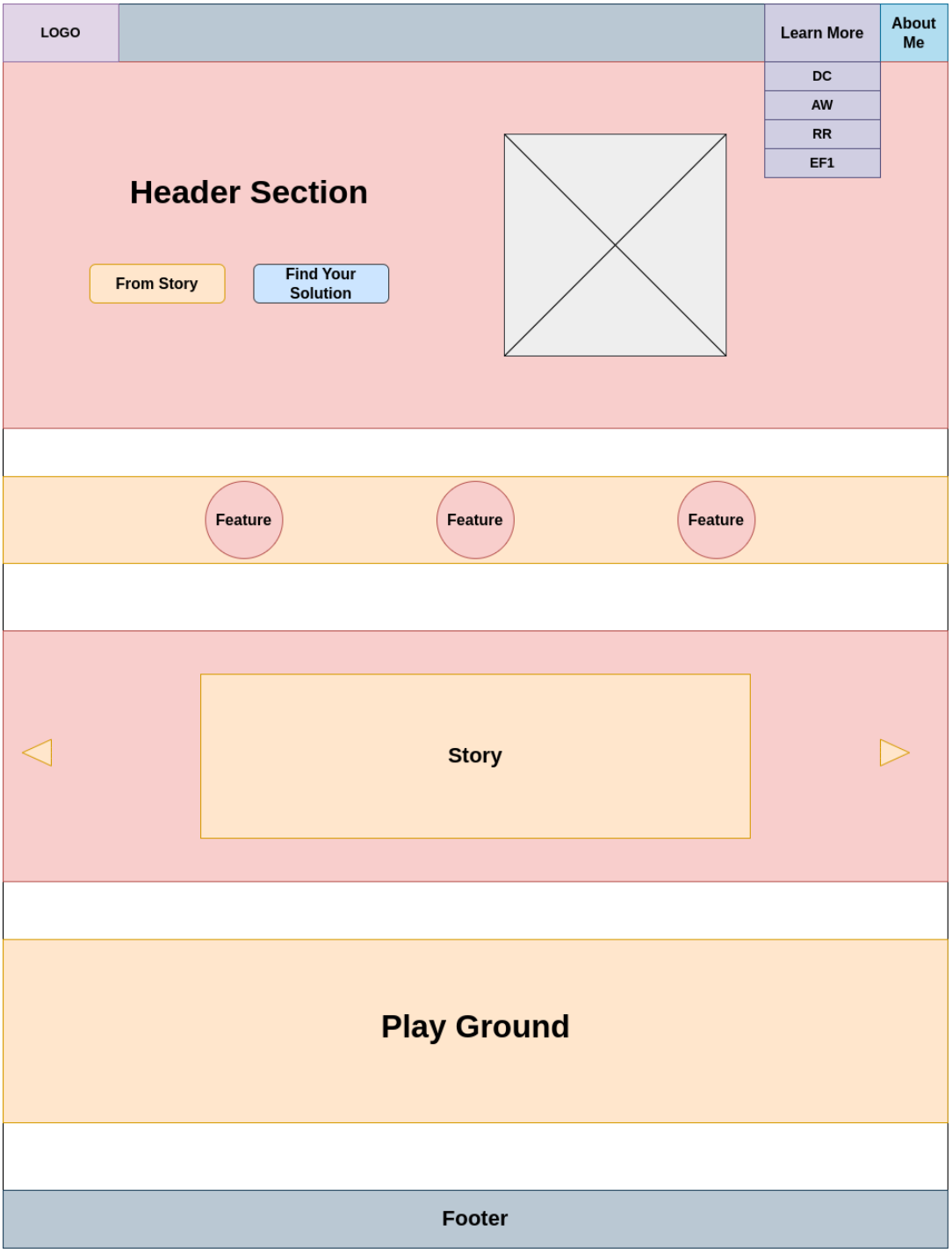


Fig. 4.1: Wire frame of Fairallol

Technology used in the website Fairallol

The name of the website is Fairallol[3], be built using the Vue framework, with technologies such as HTML, CSS, JavaScript. Bootstrap5 is used as the component library to glorify the website, and NerdFonts Icon provides some adorable icons. Axios sends requests to communicate with the back-end server, receives responses and handles errors.

The backend service is written in the Go language web framework *Gin*, two replicated classical implementations (DC and RR) and two algorithms based on AW and EF are used as services.

All front-end and back-end codes was produced by the author. [4] [5]

Website Content Design

Fairallol is designed as a new generation of innovative fair allocation website. Its interface is interactive, engaging, and educational. The website attempts to use vivid examples and interaction with participants to awaken their thinking and understanding of fair allocation issues, and to provide practical convenience in their lives.

Therefore, Fairallol's content is well designed, easy-to-read, engaging and inspiring, with easy-to-understand explanations of each algorithm in a beginner-friendly manner, and a corresponding playground for visitors to play with. At the same time, the functionality is robust and the user experience is user-friendly.

Evaluation

- Examine whether it contains a complete proof and the necessary explanation.
- Create an implementation analysis of the method and evaluate its usefulness.
- Analyze the web's usability and operation. (Is it bug-free and easy to use)

Minor Testing

A small pre-launch test should be conducted before the Fairallol is released, and the website should be distributed to a small group of people to test the robustness, functionality, and ease of use of the UI, etc.

[3] Fair Allocation LOL

[4] For front-end code, please move to: https://github.com/yongtenglei/fairallol

[5] For the back-end code, please move to: https://github.com/yongtenglei/fairallol_server

4.3 Connection between Theoretical and Empirical Methods

The process of achieving fair allocation for indivisible goods is full of challenges. A comprehensive literature review is essential since it allows for the picking and implementation of appropriate algorithms without focusing too much on strict proofs or digging into major application settings. The interrelationships of multiple fairness principles, such as Envy-Free (EF), Envy-Free up to any goods (EFx), Envy-Free up to one good (EF1), and Proportionality up to One Good (Prop1), can be properly understood by thoroughly investigating them (Table. 4.2).

Table 4.2: The relationship between various fairness criteria

	Existence		Computation	
	Without PO	With PO	Without PO	With PO
EF	No	No	NP-hard	NP-hard
EFx	Open	Open	Open	Open
EF1	Yes	Yes	Polytime	Open
Prop1	Yes	Yes	Polytime	Polytime

This comprehensive examination of fairness principles not only reduces overall workload but also provides significant insights into the specific criteria met by each deployed algorithm. As a result, the investigator can focus more effectively on improving algorithms and assessing their effectiveness. The improved understanding of fairness criteria resulting from the literature research serves as a foundation for future algorithmic development and guarantees that the fairness criteria are appropriately reflected in the approaches adopted.

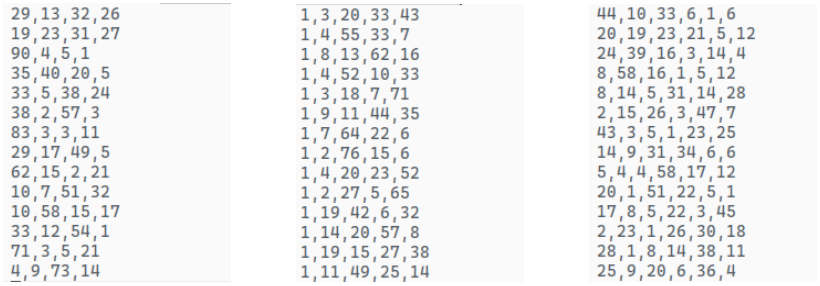
Chapter 5

Experiments

5.1 Data Preparation

As mentioned earlier, since the real data are absent, complete simulated data will be used to test the algorithm (in fact, this provides a wider coverage than the real data). Please refer to section 4.2.1 for the procedure to generate the data.

Use Python to generate the data and store it as a plain text file for reading by the Go language test program.



```
29,13,32,26
19,23,31,27
90,4,5,1
35,40,20,5
33,5,38,24
38,2,57,3
83,3,3,11
29,17,49,5
62,15,2,21
10,7,51,32
10,58,15,17
33,12,54,1
71,3,5,21
4,9,73,14
```

(a) 4_Normal_partial

```
1,3,20,33,43
1,4,55,33,7
1,8,13,62,16
1,4,52,10,33
1,3,18,7,71
1,9,11,44,35
1,7,64,22,6
1,2,76,15,6
1,4,20,23,52
1,2,27,5,65
1,19,42,6,32
1,14,20,57,8
1,19,15,27,38
1,11,49,25,14
```

(b) 5_Similar_partial

```
44,10,33,6,1,6
20,19,23,21,5,12
24,39,16,3,14,4
8,58,16,1,5,12
8,14,5,31,14,28
2,15,26,3,47,7
43,3,5,1,23,25
14,9,31,34,6,6
5,4,4,58,17,12
20,1,51,22,5,1
17,8,5,22,3,45
2,23,1,26,30,18
28,1,8,14,38,11
25,9,20,6,36,4
```

(c) 6_Tie_partial

Fig. 5.1: Data preparation

5.2 Divide-and-Choose

The Divide and Choose algorithm is a sensible and intuitive method for allocating indivisible items between two agents. The "divider," a single agent, divides the things into two groups based on its valuations. The other agent, the "chooser," then chooses its preferred group, leaving the remaining group to the divider. **This method ensures envy-freeness, as the chooser gets the group they value most, while the divider, who made the par-**

tition, believes both groups are basically equal valuable. Divide and Choose is very beneficial in resolving disputes and allocating resources since it ensures a fair solution for both parties.

5.2.1 Pseudocode

1. First randomly divide the two participants into “divider” and “chooser”.
2. The only way to ensure maximum utility is for the “divider” to allocate the two groups as evenly as possible.
3. Uses bit-masking to efficiently traverse all cases and divide the items into the best group A and the best group B.
4. The “chooser” chooses the group with the greatest utility to itself, leaving the other part to the “divider”.

5.2.2 Optimization

```
// Iterate through all possible item combinations using bitmasks
for i := 0; i < (1 << len(items)); i++ {
    group1 := []string{}
    group2 := []string{}
    valueGroup1 := 0
    valueGroup2 := 0

    for j, item := range items {
        if i&(1<<j) != 0 {
            group1 = append(group1, item.Name)
            valueGroup1 += divider.Valuations[item.Name]
        } else {
            group2 = append(group2, item.Name)
            valueGroup2 += divider.Valuations[item.Name]
        }
    }

    // update the best Group
    difference := int(math.Abs(float64(valueGroup1 - valueGroup2)))
    if difference < minDifference {
        minDifference = difference
        bestGroup1 = group1
        bestGroup2 = group2
    }
}
```

Fig. 5.2: Iterate through efficiently all possible item combinations using bit-masks

The algorithm uses bit masks to perform bitwise operations to combine the possibilities of all item combinations, which is one of the fastest implementations and greatly improves the performance of the algorithm.

5.2.3 Normal Case and Similar Case

Table 5.1: Normal Case and Similar Case of DC $N = 5$

Normal Case (DC)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	12	37	16	22	13	50
Bob	2	17	27	16	38	55

Similar Case (DC)					
Item1	Item2	Item3	Item4	Item5	Utility
1	26	30	31	12	44
1	25	38	18	18	63

As in the algorithm procedure, the algorithm averages the utility of the two groups as much as possible by means of a bit-masking operation. Using the case of 5 items as a demonstration, it can be observed that the algorithm can always find the item with the *lowest utility* and add it to the otherwise relatively *inferior* side.

Table 5.2: The "divider" maintains a balance between the two groups of items

Divide And Choose (DC)						
Utility of two groups of items (Alice's perspective) in Normal Case						
Items	Item1	Item2	Item3	Item4	Item5	Utility Gained
Group1	12	0	16	22	0	50
Group2	0	37	0	0	13	50
Difference						0

In the Normal Case example, the algorithm appoints Alice as "Divider" and divides the items into two groups as evenly as possible from its own viewpoint, so that the difference in utility between the two groups is as small as possible (after a bitmask operation, the difference in utility between the two groups is 0 in this example). Then "Selector" Bob chooses the part it prefers.

5.2.4 Tie-preference Case

Table 5.3: Tie Case for DC $N = 5$

Tie Case (DC)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	5	5	15	14	61	39
Bob	5	5	15	14	61	61

The algorithm seems to have encountered a strong challenger. Not only does it encounter the exact same preference input, but it also encounters an item that has an overwhelming advantage over other items. However, DC does a good job of not sticking to an average in the number of items allocated, but rather an average in value, which makes sense in this respect.

Shallow test evidence see Appendix A.1(a)

5.2.5 Overall Testing

```
➜ fairallol_server/allocations/divideChoose main ✓ go test -v
=== RUN   TestDivideAndChoose
===============Test Divide and Choose=================
Test for:
        N: 4, Pattern: normal, SampleNum: 1000, FileName: 4_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 20.26, Time elapsed: 8.732474ms, Average Goods Diff: 1.16

Test for:
        N: 4, Pattern: similar, SampleNum: 100, FileName: 4_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 19.12, Time elapsed: 10.143776ms, Average Goods Diff: 1.15

Test for:
        N: 4, Pattern: tie, SampleNum: 500, FileName: 4_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 19.62, Time elapsed: 2.517454ms, Average Goods Diff: 1.12

Test for:
        N: 5, Pattern: normal, SampleNum: 1000, FileName: 5_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 20.98, Time elapsed: 11.07734ms, Average Goods Diff: 1.70

Test for:
        N: 5, Pattern: similar, SampleNum: 100, FileName: 5_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 17.85, Time elapsed: 10.768934ms, Average Goods Diff: 1.72

Test for:
        N: 5, Pattern: tie, SampleNum: 500, FileName: 5_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 21.01, Time elapsed: 5.192414ms, Average Goods Diff: 1.71

Test for:
        N: 6, Pattern: normal, SampleNum: 1000, FileName: 6_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 19.90, Time elapsed: 25.108888ms, Average Goods Diff: 1.76

Test for:
        N: 6, Pattern: similar, SampleNum: 100, FileName: 6_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 17.45, Time elapsed: 27.226924ms, Average Goods Diff: 1.72

Test for:
        N: 6, Pattern: tie, SampleNum: 500, FileName: 6_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 19.01, Time elapsed: 12.288562ms, Average Goods Diff: 1.60

--- PASS: TestDivideAndChoose (0.12s)
PASS
ok      rey.com/fairallol/allocations/divideChoose      0.119s
```

Fig. 5.3: Overall Test for DC (Each pattern 500 test cases)

Table 5.4: Testing for DC

Divide And Choose (DC)									
N	4			5			6		
Pattern	Normal	Similar	Tie	Normal	Similar	Tie	Normal	Similar	Tie
Avg. $Score_{diff}$	20.26	19.12	19.62	20.98	17.85	21.01	19.90	17.45	19.01
Avg. $Goods_{diff}$	1.16	1.15	1.12	1.70	1.72	1.71	1.76	1.72	1.60
Time Elapsed	8.732474ms	10.143776ms	2.517454ms	11.07734ms	10.768934ms	5.192414ms	25.108888ms	27.226924ms	12.288562ms

The algorithm was evaluated, by three different N, and three different input preference patterns, by 500 test cases, to obtain the above charts. These charts will be discussed in detail as evidence in Section.5.6.2.

5.3 Adjusted-Winner

The Adjusted Winner algorithm is commonly used in the fair allocation of indivisible items. This algorithm seeks to allocate limited resources to participants while satisfying fairness principles as much as possible. This is achieved by assigning items to participants based on the value of the winner, which is determined by their relative preference for the items. To ensure fairness, the algorithm uses an adjustment factor that reflects the impact of the assigned items on the remaining participants and adjusts their valuations accordingly. In this implementation

$$af = adjusted\ factor = sum\ of\ points\ /\ number\ of\ agents.$$

And,

*adjusted valuation -= af * the number of items this participant has been (pre)-allocated.*

This algorithm guarantees two fairness principles: "independence" and "envy-freeness". Independence means that the algorithm recalculates each participant's score and each item's valuation before each allocation and adjusts each participant's score after each allocation based on the effect of the allocated items on the remaining participants. Fairness is ensured in the final allocation always implies that no other participant's allocation has a higher utility than their existing allocation. In other words, no one is envious of others.

5.3.1 Algorithm procedure

1. Calculate the adjusted factor.
2. Allocate each item to a participant based on the evaluations adjusted by the adjusted factor.
3. Check if the allocations are envy-free, if yes, the allocation is finished and the PO is satisfied as much as possible. If no, go to the next step.
4. Unallocate all items that have been assigned to participants who failed the envy-free check, and proceed to the allocation phase again.

5.3.2 Optimization

Unfortunately, AW does not always find a fair allocation if a fixed adjustment factor is used. This implementation borrows the idea of simulated annealing, so that after a certain number of iterations, the adjustment factor is still not found to be allocated, and then a small adjustment is made to achieve algorithmic completeness (all items should be allocated to all participants), which is an algorithmic compromise.

```
// The adjustment factor = sum of points / number of agents.
adjustmentFactor := float64(totalPoints) / float64(len(agents))
if adjustmentFactor <= 0 {
    panic("adjustmentFactor <= 0")
}

allocatedItems := make(map[string]bool)

// Shuffle the order of the items
rand.Shuffle(len(items), func(i, j int) {
    items[i], items[j] = items[j], items[i]
})

counter := 1
for {
    if counter > threshold {
        rate := float64(threshold) / float64(counter)
        adjustmentFactor *= rate
    }

    // Shuffle the order of the agents
    rand.Shuffle(len(agents), func(i, j int) {
        agents[i], agents[j] = agents[j], agents[i]
    })

    allocateItemsToAgents(agents, items, allocatedItems, adjustmentFactor)

    done := checkAndReallocateItems(agents, items, allocatedItems)
```

Fig. 5.4: The idea of simulated annealing in algorithm for completeness compromise

The use of a shuffling mechanism also helps to avoid possible biases in the allocation process.

5.3.3 Normal Case and Similar Case

Table 5.5: Normal Case and Similar Case of AW $N = 5$

Normal Case (AW)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	12	37	16	22	13	71
Bob	2	17	27	16	38	65

Similar Case (AW)					
Item1	Item2	Item3	Item4	Item5	Utility
1	26	30	31	12	58
1	25	38	18	18	56

In normal preference pattern as well as in similar preference pattern, AW can work appropriately. Even achieved a greater social welfare [1] than DC . The algorithm first allocates

[1] social welfare here means the sum of utility obtained by all participants

items to participants according to their highest valuation (satisfying PO as much as possible), and then checks whether the allocation is envy-free, and ends the allocation when everyone is not envy of others' allocation.

5.3.4 Tie-preference Case

Table 5.6: Tie Case for AW $N = 5$

Tie Case (AW)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	5	5	15	14	61	75
Bob	5	5	15	14	61	25

When it comes to preferences for Tie pattern, Adjusted Winner (AW) algorithm may encounter some difficulties compared to other algorithms such as DC. AW assigns items to participants based on their highest evaluation, but it faces a Tie input pattern, and can only temporarily assign participants in a certain order. The envy-free process may also get stuck in a "hard-to-break-through situation" if a fixed adjustment factor is used. To ensure algorithmic completeness, the algorithm enters an annealing phase after a certain iteration threshold is reached, where the adjustment factor decreases progressively and the deadlock begins to break. However, during this phase, the allocation can become random and really dependent on the participants' evaluation of the items. Therefore, AW may not be as effective as DC when it comes to handling preferences for Tie pattern.

Shallow test evidence see Appendix A.1(b)

5.3.5 Overall Testing

```
➜ fairallol_server/allocations/adjustedWinner main ✓ go test -v
=== RUN   TestAdjustedWinner
===============Test Adjusted Winner=================
Test for:
        N: 4, Pattern: normal, SampleNum: 1000, FileName: 4_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 23.46, Time elapsed: 59.941853ms, Average Goods Diff: 0.05

Test for:
        N: 4, Pattern: similar, SampleNum: 100, FileName: 4_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 30.75, Time elapsed: 15.283481ms, Average Goods Diff: 0.02

Test for:
        N: 4, Pattern: tie, SampleNum: 500, FileName: 4_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 24.73, Time elapsed: 51.420044ms, Average Goods Diff: 0.15

Test for:
        N: 5, Pattern: normal, SampleNum: 1000, FileName: 5_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 24.60, Time elapsed: 48.398528ms, Average Goods Diff: 1.00

Test for:
        N: 5, Pattern: similar, SampleNum: 100, FileName: 5_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 30.71, Time elapsed: 29.986243ms, Average Goods Diff: 1.00

Test for:
        N: 5, Pattern: tie, SampleNum: 500, FileName: 5_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 27.45, Time elapsed: 24.753526ms, Average Goods Diff: 1.00

Test for:
        N: 6, Pattern: normal, SampleNum: 1000, FileName: 6_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 20.88, Time elapsed: 117.58782ms, Average Goods Diff: 0.07

Test for:
        N: 6, Pattern: similar, SampleNum: 100, FileName: 6_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 22.83, Time elapsed: 76.304055ms, Average Goods Diff: 0.02

Test for:
        N: 6, Pattern: tie, SampleNum: 500, FileName: 6_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 21.22, Time elapsed: 52.306141ms, Average Goods Diff: 0.09

--- PASS: TestAdjustedWinner (0.48s)
PASS
ok      rey.com/fairallol/allocations/adjustedWinner    0.483s
```

Fig. 5.5: Overall Test for AW (Each pattern 500 test cases)

Table 5.7: Testing for AW

Adjusted Winner (AW)									
N	4			5			6		
Pattern	Normal	Similar	Tie	Normal	Similar	Tie	Normal	Similar	Tie
Avg. $Score_{diff}$	23.46	30.75	24.73	24.60	30.71	27.45	20.88	22.83	21.22
Avg. $Goods_{diff}$	0.05	0.02	0.15	1.00	1.00	1.00	0.07	0.02	0.09
Time Elapsed	59.941853ms	15.283481ms	51.420044ms	48.398528ms	29.986243ms	24.753526ms	117.58782ms	76.304055ms	52.306141ms

The algorithm was evaluated, by three different N, and three different input preference patterns, by 500 test cases, to obtain the above charts. These charts will be discussed in detail as evidence in Section.5.6.2.

5.4 Round-Robin

The Round Robin (RR) algorithm is another widely used algorithm for the allocation of indivisible goods. The core idea is to achieve fair allocation by circularly allocating an item to each participant in a predetermined order. Specifically, the algorithm iterates through the participants in a loop and allocates the most preferred items as possible according to the participants' evaluations. Since participants receive at least one item in each loop, it is easy to balance the number of items allocated.

The RR algorithm naturally satisfies EF1 because the algorithm works in such a way that each participant is allocated an item that is at most one worse than the one that is preferred less. Similarly, it also satisfies independence, as the allocation of items depends only on the participants' evaluation of the items and the sequencing mechanism, independent of the behavior of other participants.

5.4.1 Algorithm procedure

1. Queue participants in a certain order or shuffle them.
2. Iterate participants in order and allocate their favorite item as much as possible.
3. Continue until all items have been allocated. This allocation is independent and EF1.

5.4.2 Normal Case and Similar Case

Table 5.8: Normal Case and Similar Case of RR $N = 5$

Normal Case (RR)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	12	37	16	22	13	71
Bob	2	17	27	16	38	65

Similar Case (RR)					
Item1	Item2	Item3	Item4	Item5	Utility
1	26	30	31	12	58
1	25	38	18	18	56

It is very easy to see that the order starts with Alice, then each person is assigned their current favored item, until all items are allocated.

Table 5.9: Round Robin allocation process in Normal Case

Rround Robin (RR)						
Items	Item1	Item2	Item3	Item4	Item5	Participator
1		37				Alice
2					38	Bob
3		37		22		Alice
4			27		38	Bob
5	12	37		22		Alice

Again using the Normal Case as an example, the algorithm starts with Alice (in fact the order is random, Bob may need more luck), choosing the unassigned item that he likes best. Then it's Bob's turn to choose, and so on... until all items have been allocated to all participants.

5.4.3 Tie-preference Case

Table 5.10: Tie Case for RR $N = 5$

Tie Case (RR)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	5	5	15	14	61	80
Bob	5	5	15	14	61	20

Similarly, RR, like AW, does not take the Tie input pattern well into account for DC. When in a Tie input pattern, the first participant to be allocated will have a huge advantage, since it will always have access to the item that is acknowledged to be the most valuable. If this most valuable item has an overwhelming dominance in the evaluation,

for example, a house versus a pile of miscellaneous items, using RR will result in the first assigned participant gaining an absolute numerical lead. Certainly, such a setting would never be possible in real life using the RR algorithm.

Shallow test evidence see Appendix A.1(c)

5.4.4 Overall Testing

```
➜ fairallol_server/allocations/roundRobin main ✓ go test -v
=== RUN   TestRoundRobin
===============Test roundRobin=================
Test for:
        N: 4, Pattern: normal, SampleNum: 1000, FileName: 4_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 19.15, Time elapsed: 1.474912ms, Average Goods Diff: 0.00

Test for:
        N: 4, Pattern: similar, SampleNum: 100, FileName: 4_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 27.07, Time elapsed: 1.201002ms, Average Goods Diff: 0.00

Test for:
        N: 4, Pattern: tie, SampleNum: 500, FileName: 4_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 31.34, Time elapsed: 1.636466ms, Average Goods Diff: 0.00

Test for:
        N: 5, Pattern: normal, SampleNum: 1000, FileName: 5_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 20.47, Time elapsed: 2.221386ms, Average Goods Diff: 1.00

Test for:
        N: 5, Pattern: similar, SampleNum: 100, FileName: 5_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 25.57, Time elapsed: 2.911443ms, Average Goods Diff: 1.00

Test for:
        N: 5, Pattern: tie, SampleNum: 500, FileName: 5_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 29.80, Time elapsed: 1.991294ms, Average Goods Diff: 1.00

Test for:
        N: 6, Pattern: normal, SampleNum: 1000, FileName: 6_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 15.36, Time elapsed: 3.082858ms, Average Goods Diff: 0.00

Test for:
        N: 6, Pattern: similar, SampleNum: 100, FileName: 6_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 19.46, Time elapsed: 3.014712ms, Average Goods Diff: 0.00

Test for:
        N: 6, Pattern: tie, SampleNum: 500, FileName: 6_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 24.96, Time elapsed: 3.434005ms, Average Goods Diff: 0.00

--- PASS: TestRoundRobin (0.02s)
PASS
ok      rey.com/fairallol/allocations/roundRobin        0.027s
```

Fig. 5.6: Overall Test for RR (ach pattern 500 test cases)

Table 5.11: Testing for RR

Round Robin (RR)									
N	4			5			6		
Pattern	Normal	Similar	Tie	Normal	Similar	Tie	Normal	Similar	Tie
Avg. Score_{diff}	19.15	27.07	31.34	20.47	25.57	29.80	15.36	19.46	24.96
Avg. Goods_{diff}	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00
Time Elapsed	1.474912ms	1.201002ms	1.636466ms	2.221386ms	2.911443ms	1.991294ms	3.082858ms	3.014712ms	3.434005ms

The algorithm was evaluated, by three different N, and three different input preference patterns, by 500 test cases, to obtain the above charts. These charts will be discussed in detail as evidence in Section.5.6.2.

5.5 Envy-fairness

The Envy-Free up to one item (EF1) algorithm is a compromise of Envy-Free (EF). This means that when a envy occurs, the envy disappears after the envied person removes an item, and such a scenario is still considered to satisfy an envy-free, namely EF1.

This implementation is one that takes into account the EF1 idea, but is more realistic. By keeping track of the number of items allocated to two participants, when either party has more than two items than the other, the items need to be exchanged to achieve fairness. Like the AW algorithm, the initial allocation will be based on the highest evaluation of the items to the corresponding participant, but it should be noted that this algorithm tends to maintain a balance in the number of items allocated, which is not quite the same as AW.

5.5.1 Algorithm procedure

1. Items are allocated to participants according to their highest evaluations.
2. When preferences are the same, allocate to the participant who has received fewer items so far.
3. When any participant receives two more items, a shift is triggered to ensure balance.
4. The weaker participant takes the most preferred item of the dominant participant, or if not, takes the least valuable item of the dominant participant.

5.5.2 Normal Case and Similar Case

Table 5.12: Normal Case and Similar Case of EF1 $N = 5$

Normal Case (EF1)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	12	37	16	22	13	71
Bob	2	17	27	16	38	65

Similar Case (EF1)					
Item1	Item2	Item3	Item4	Item5	Utility
1	26	30	31	12	58
1	25	38	18	18	56

5.5.3 Tie-preference Case

Table 5.13: Tie Case for EF1 $N = 5$

Tie Case (EF1)						
	Item1	Item2	Item3	Item4	Item5	Utility
Alice	5	5	15	14	61	81
Bob	5	5	15	14	61	19

In this implementation of the EF1 algorithm, the algorithm tries to balance the number of allocations to try to find the optimal allocation in the face of the three patterns of preferences. There are two layers of safeguards here, when allocating items with the largest evaluation of the item, if more than one participants have the same evaluation, preference is given to the participant who has received the fewest items so far. In addition, if a participant receives too many items, a transfer will be triggered to maintain the balance. Before triggering the transfer, this algorithm can be considered as a greedy algorithm that seeks to achieve maximum PO, and after triggering the transfer, this EF1 fairness criterion is guaranteed.

Shallow test evidence see Appendix A.1(d)

5.5.4 Overall Testing

```
➜ fairallol_server/allocations/saveworld main ✓ go test -v
=== RUN   TestSaveWorld
===============Test SaveWorld=================
Test for:
        N: 4, Pattern: normal, SampleNum: 1000, FileName: 4_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 20.03, Time elapsed: 3.452247ms, Average Goods Diff: 0.00

Test for:
        N: 4, Pattern: similar, SampleNum: 100, FileName: 4_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 29.76, Time elapsed: 3.218606ms, Average Goods Diff: 0.00

Test for:
        N: 4, Pattern: tie, SampleNum: 500, FileName: 4_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 19.61, Time elapsed: 1.371437ms, Average Goods Diff: 0.00

Test for:
        N: 5, Pattern: normal, SampleNum: 1000, FileName: 5_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 20.36, Time elapsed: 3.877644ms, Average Goods Diff: 1.00

Test for:
        N: 5, Pattern: similar, SampleNum: 100, FileName: 5_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 27.71, Time elapsed: 2.510686ms, Average Goods Diff: 1.00

Test for:
        N: 5, Pattern: tie, SampleNum: 500, FileName: 5_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 22.27, Time elapsed: 1.188571ms, Average Goods Diff: 1.00

Test for:
        N: 6, Pattern: normal, SampleNum: 1000, FileName: 6_data_normal_1000.txt, FileDataLen: 1000
Average Scores Diff: 16.30, Time elapsed: 3.011717ms, Average Goods Diff: 0.00

Test for:
        N: 6, Pattern: similar, SampleNum: 100, FileName: 6_data_similar_100.txt, FileDataLen: 1000
Average Scores Diff: 22.28, Time elapsed: 3.505614ms, Average Goods Diff: 0.00

Test for:
        N: 6, Pattern: tie, SampleNum: 500, FileName: 6_data_tie_500.txt, FileDataLen: 500
Average Scores Diff: 16.05, Time elapsed: 2.104929ms, Average Goods Diff: 0.00

--- PASS: TestSaveWorld (0.03s)
PASS
ok      rey.com/fairallol/allocations/saveworld 0.031s
```

Fig. 5.7: Overall Test for EF1 (Each pattern 500 test cases)

Table 5.14: Testing for EF1

Envy-fairness (EF1)									
N	4			5			6		
Pattern	Normal	Similar	Tie	Normal	Similar	Tie	Normal	Similar	Tie
Avg. ScoreDiff	20.03	29.76	19.61	20.36	27.71	22.27	16.30	22.28	16.05
Avg. GoodsDiff	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00
Time Elapsed	3.452247ms	3.218606ms	1.371437ms	3.877644ms	2.510686ms	1.188571ms	3.011717ms	3.505614ms	2.104929ms

The algorithm was evaluated, by three different N, and three different input preference patterns, by 500 test cases, to obtain the above charts. These charts will be discussed in detail as evidence in Section.5.6.2.

5.6 Overall Analysis

5.6.1 External Testing (Fairallol Vs. Fairpy)

In external tests, Fairallol will be tested with Fairpy in both shallow and in-depth tests, using the Round Robin algorithm. In the shallow test, it is used to verify that the behavior of the two fair allocation libraries is consistent.

```
similar:
 Alice gets {Item1,Item2,Item4} with value 58.
Bob gets {Item3,Item5} with value 56.

elapsed time:  0.0002803802490234375

normal:
 Alice gets {Item1,Item2,Item4} with value 71.
Bob gets {Item3,Item5} with value 65.

elapsed time:  0.0001437664031982422

tie:
 Alice gets {Item2,Item4,Item5} with value 80.
Bob gets {Item1,Item3} with value 20.

elapsed time:  0.0001392364501953125
```

Fig. 5.8: Shallow test for Fairpy (the Same behavior as Fairallol)

In the deep tests, Fairpy will be tested with the same set of measures as Fairallol, but with a single focus on the runtime metric (since the behavior of both libraries is consistent regrading to RR algorithm)

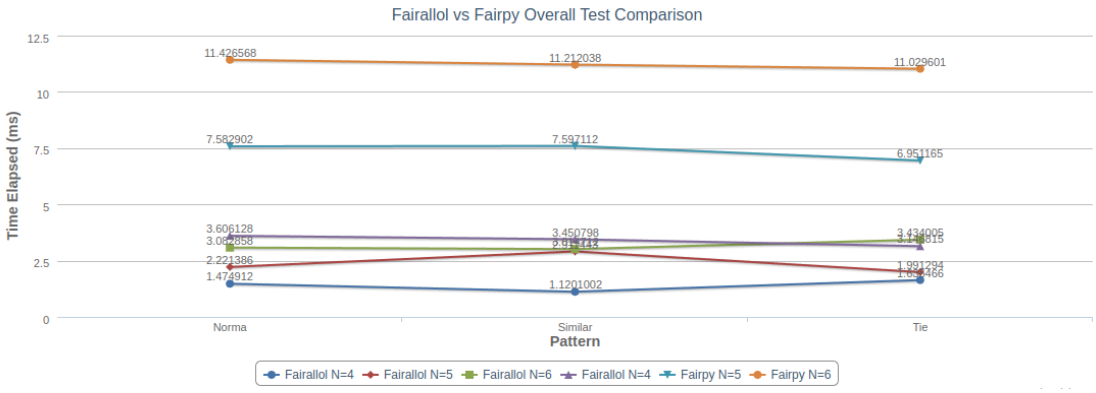


Fig. 5.9: Fairallol behaves the same as Fairpy and runs nearly three times faster.

Evidence of in-depth tests for Fairpy see Appendix A.2

After testing Fairpy with the same test set as Fairallol (each preference input pattern with 500 cases). **It is found that Fairallol not only has the same behavior as Fairpy, but also has a speedup of nearly 3 times**.

5.6.2 Internal Testing (Fairallol)

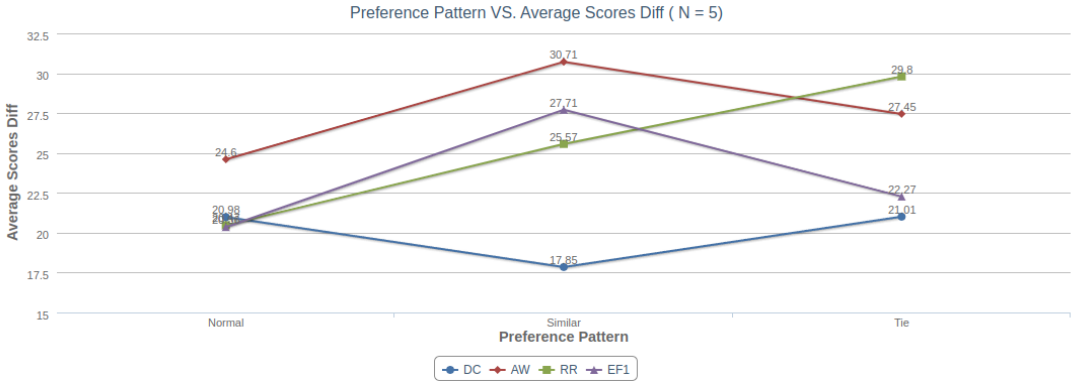


Fig. 5.10: Preference Pattern VS. Average Scores Diff (N = 5)

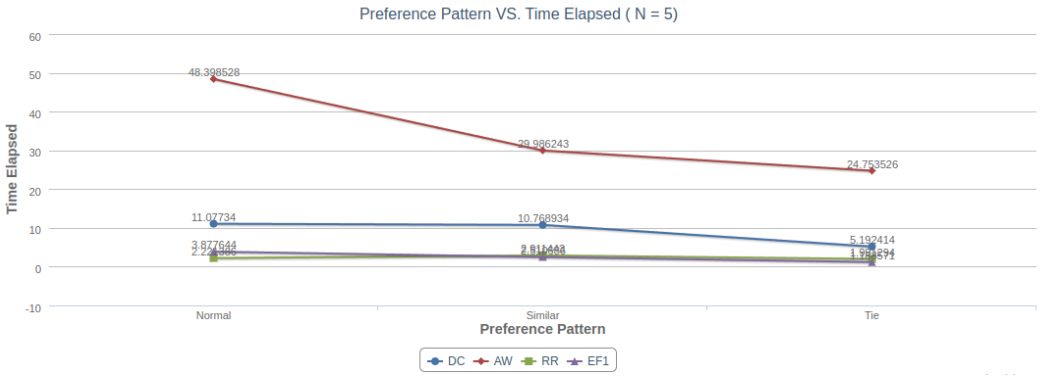


Fig. 5.11: Preference Pattern VS. Time Elapsed (N = 5)

For more evidence, please refer to Appendix B

The Round Robin algorithm can be conceived as the simplest but also has additional realistic implications as a fairness algorithm. The simplicity of the implementation allows it to have essentially the highest running efficiency, and to imply certain fairness criteria EF1 , etc. However, it is worth noting that RR's allocation prefers its own mechanism rather than the settings with the allocated items or the evaluation of the items by the participants, leading it to fall short of the fairness criteria when dealing with non-general patterns of preferences (*Similar* and *Tie*). The other three algorithms, on the other hand,

suffer from a certain amount of confusion in the *Similar* pattern.

Adjusted Winner (AW) seems to have the worst performance, the most time consuming and the difference in the obtained utility seems to be large. The reason for the long time consumption may be due to the fact that a fixed adjustment factor is used upfront, and if a reasonable allocation is not achieved in the first few attempts, the algorithm must iterate to a certain threshold before the adjustment factor is annealed, which is not easy to determine, and in the current implementation, it is hard-coded to simulate annealing at n=1000. This value may not be applicable in the test set. The reason for the large error may be that AW really considers envy-free in some other sense, rather than minimizing the difference between the values of the allocated items, and all the final allocations it produced are envy-free, which is missing in some implemented algorithms.

EF1 does not seem very bad, but in fact it is a greedy algorithm most of the time. Although the EF1 algorithm guarantees that each participant will receive one of his or her favorite items, it does not guarantee that the allocation is Pareto optimal. This is because the EF1 algorithm only considers the individual preferences of each participant, and does not consider the efficiency and overall welfare of the entire allocation scheme. Therefore, even if each participant is satisfied under the EF1 algorithm, the overall allocation scheme may not be optimal for all participants. Therefore, in practical applications, it is necessary to choose the appropriate fairness criterion for the allocation of indivisible items according to the specific situation.

Compared to the EF1 algorithm, the Divide And Choose (DC) algorithm is more global in nature and its idea is very simple yet useful. If there are only two participants involved in allocating items fairly, then the simplest and most effective way is “you divide and I choose”. To maximize the benefits for both parties or to achieve fairness, the divider should keep the items in both groups as even as possible, because the chooser will inevitably choose the better group. The Divide And Choose algorithm is a fair and effective way to divide indivisible goods between two participants, and its simplicity makes it easy to understand and implement in real-life scenarios. However, it may not be applicable when there are more than two participants involved, or when the participants’ preferences are not well-defined. Fortunately, this is exactly the topic of this FYP.

In summary, among the four algorithms implemented, DC should be the most efficient, easy to understand, and easy to implement fair allocation algorithm in the context of the problem of fair distributing items between two people.

Chapter 6

Visualization

6.1 Web Interface (Welcome Page)

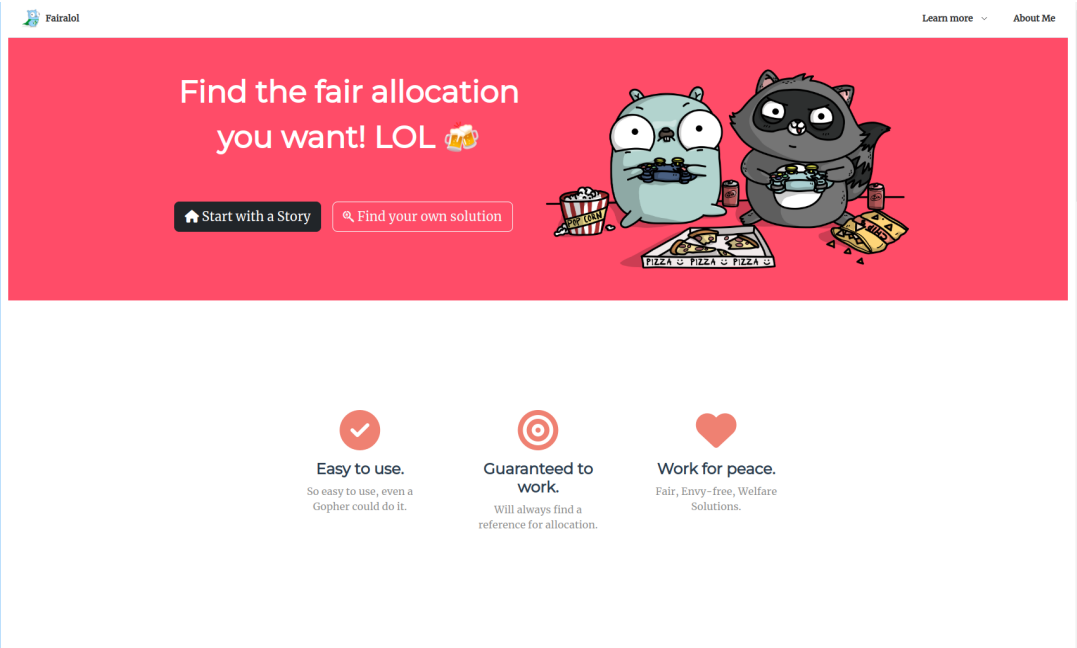


Fig. 6.1: Awesome Welcome Page

The main color of the website is red and white, providing a natural interactive experience and a cute Go language mascot, “Gopher”, to guide visitors to explore and experience the fair allocation algorithm.

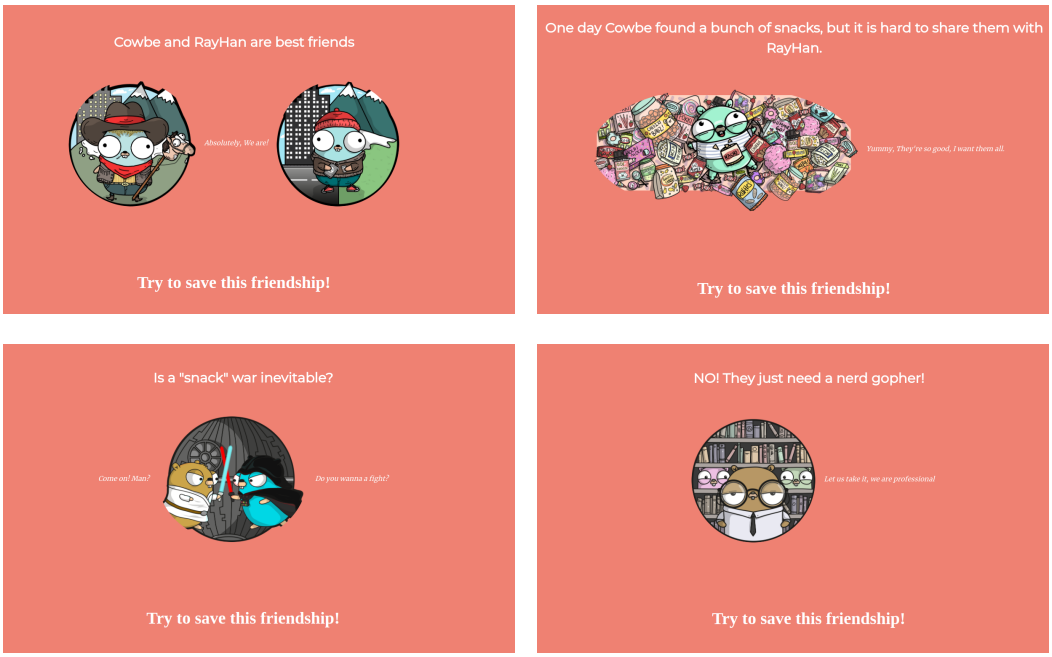


Fig. 6.2: Step into fair allocation through an interesting story

Visitors can experience fair allocation through a fun story that leads them into a scenario that requires the interaction of the allocation algorithm (See Fig.6.2).

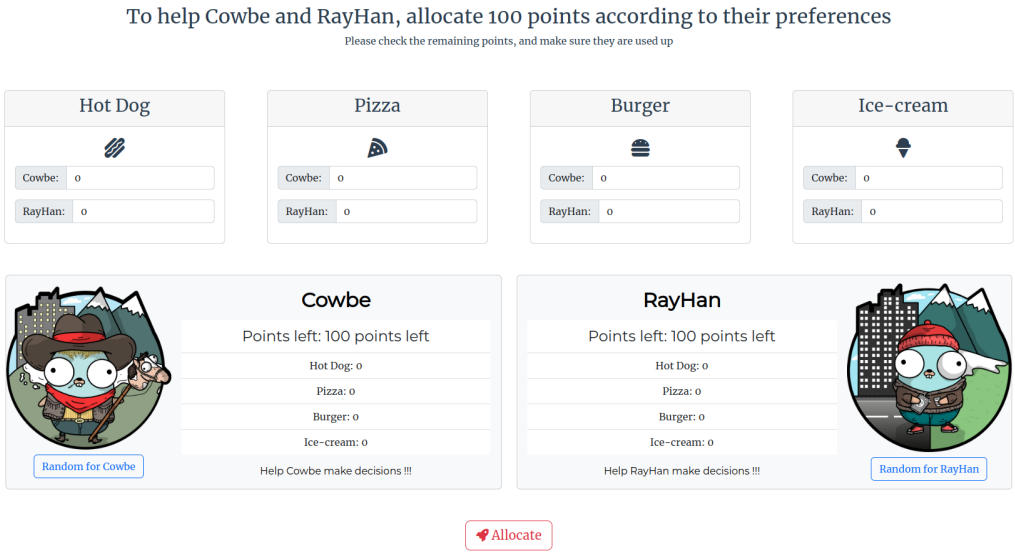


Fig. 6.3: Try the algorithm and save the friendship

After that, visitors can try out the algorithm in the playground and enter a scenario of fair allocation. Eventually find the answer to their own question (See Fig. 6.3 and Fig. 6.4).

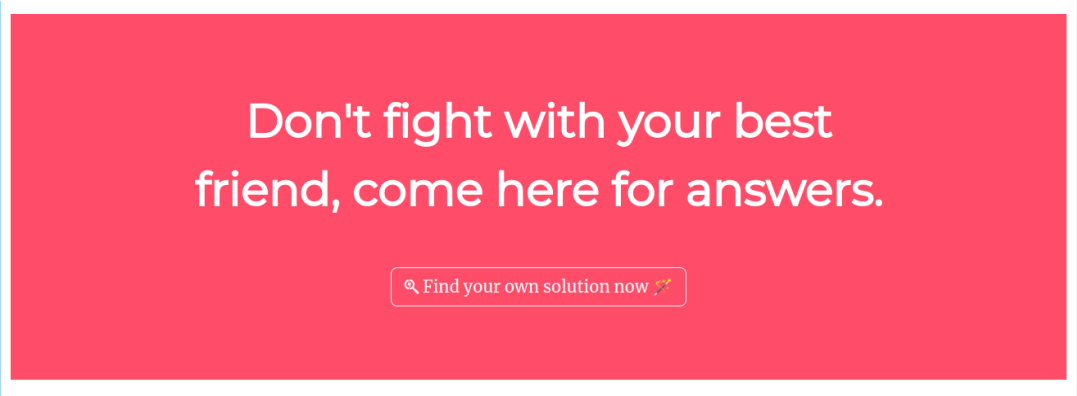


Fig. 6.4: Find your own answer and try more algorithms

6.2 Web Interface (Find Your Own Solution)

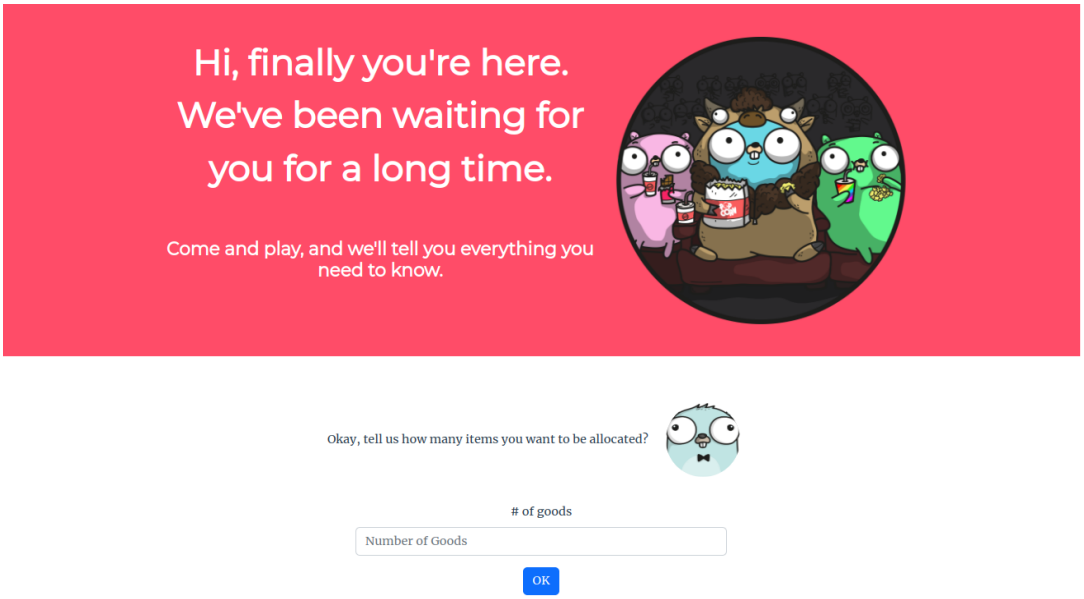
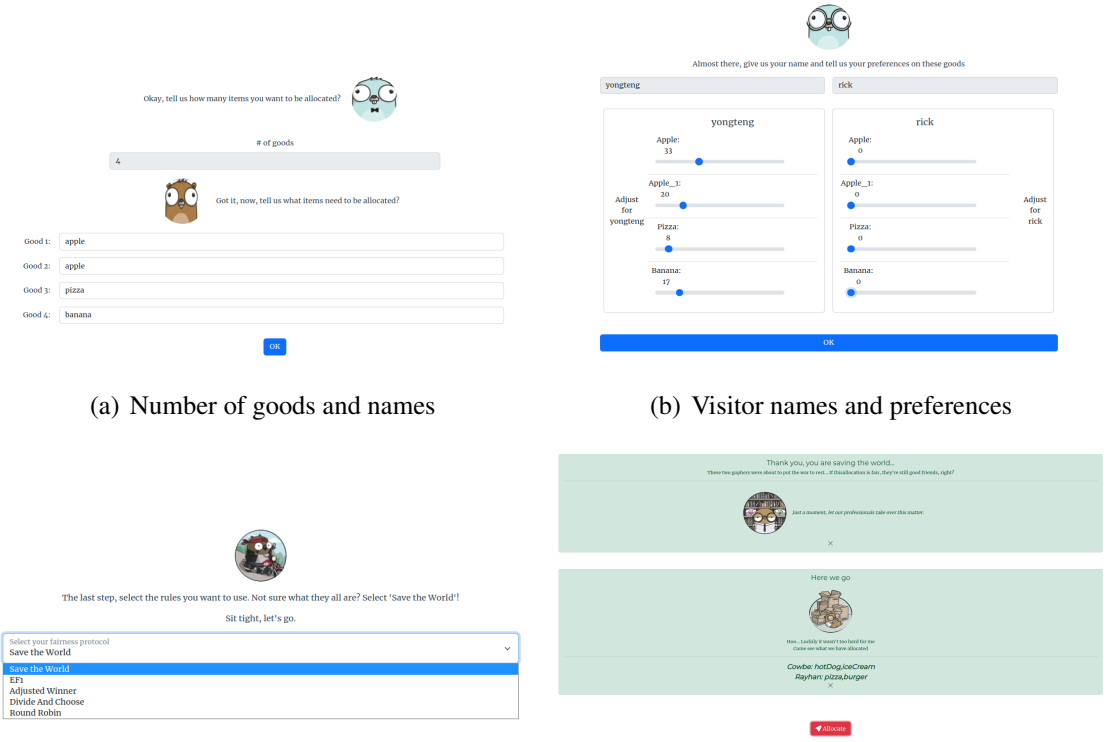


Fig. 6.5: Try more algorithms following the handy guide

Visitors can try out algorithms previously implemented in the FindSolution page and apply them to their own real-world fair-goods allocation problems. Visitors follow with the Gopher's guide to complete the form step by step, and build an allocation problem gradually, eventually finding their own reference for the answer (See Fig.6.6).



(a) Number of goods and names

(b) Visitor names and preferences

(c) Choose a algorithm want to try

(d) Got a fair allocation

Fig. 6.6: Find a allocation step by step

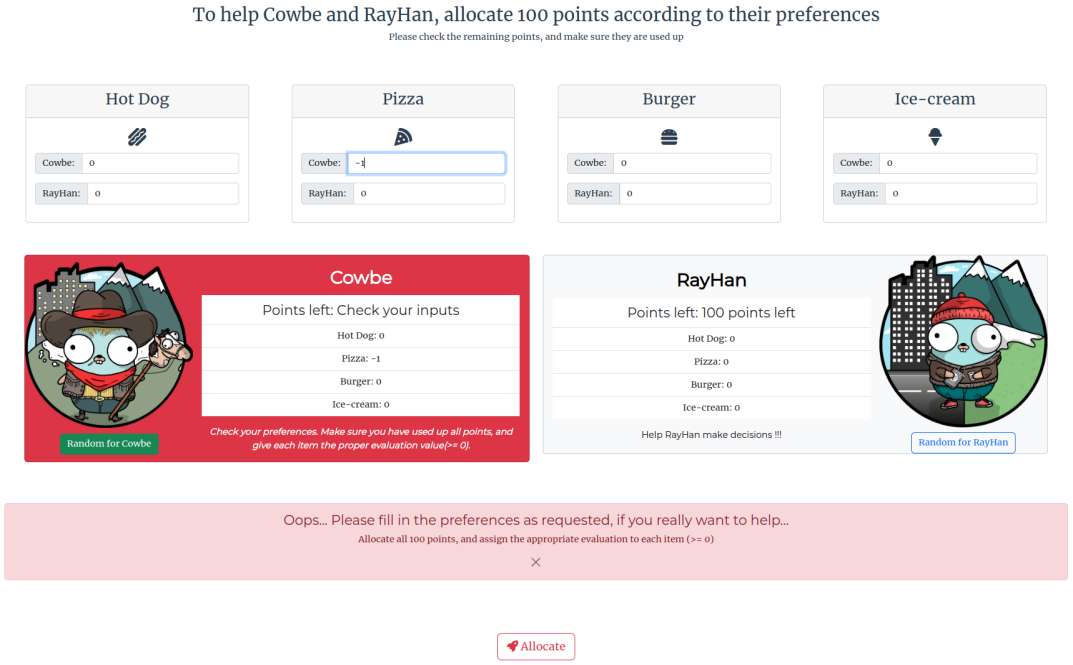
Visitors can really use their own real allocation cases to achieve different levels of fairness using different algorithms. For example, using the Round Robin algorithm, each participant has the same chance to pursue the item they want most in each round of allocation, and this outcome is only related to their own evaluation of the item and the allocation mechanism (in this case, the order of allocation). Using the Divide and Choose algorithm, a more global fairness in value can be found. With the Adjusted Winner algorithm, the fairness is obtained through some mathematical calculations. It is important to note that the Save the World algorithm is the same as the EF1 algorithm, keeping the number of items allocated as balanced as possible.

6.3 Website Robustness and User-friendliness

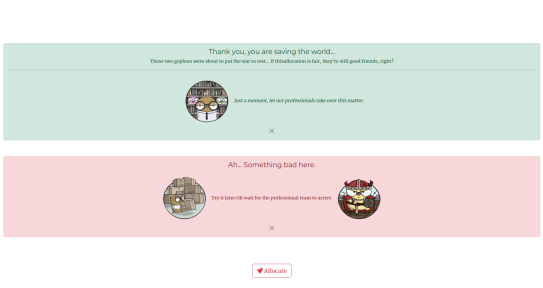
The Fairallol balance of aesthetics and functionality, while taking into account robustness and enhancing user experience.

6.3.1 Friendly Reminders

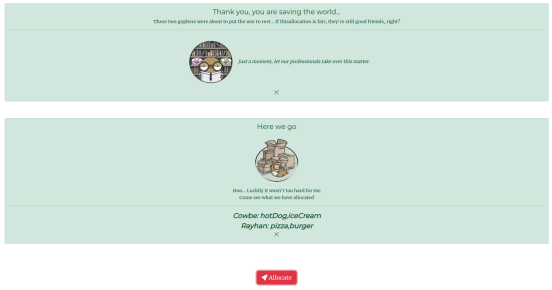
When visitors input values that are not expected or are deliberately disruptive, friendly reminders are displayed to give the necessary hints or to prevent the next step from being taken.



(a) Input illegal or all points are not used up



(b) Network errors or back-end crashes



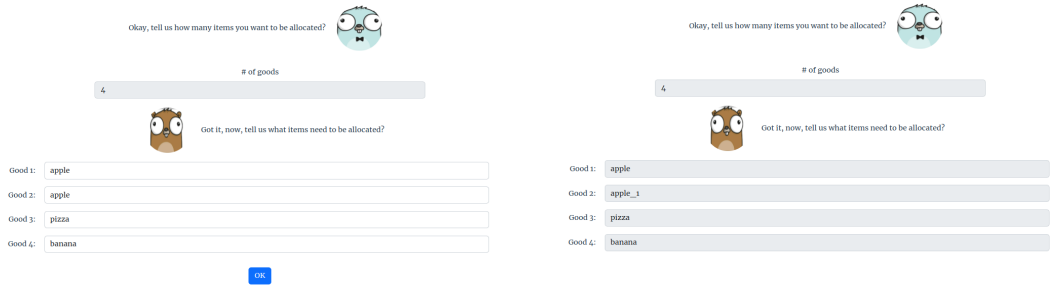
(c) Everything is going well

Fig. 6.7: Hints at the Playground

More hints example at Find Solution page see Appendix C

6.3.2 Website Robustness

The previously mentioned four algorithms are implemented as back-end services for visitors to explore and experience. However, it is required that all item names to be allocated and participant names are unique, and the front-end application ensures this for robustness.



(a) The same item name is input (b) Codes the duplicate names automatically

Fig. 6.8: The front-end program ensures that item names are not duplicated

More website robustness example at Find Solution Page see Appendix D

6.3.3 User Experience Enhancement

Recording the user's preferences for each item to be allocated is important, and is the most demanding part of the entire interaction. Random options are provided in the playground to allow visitors to experience the algorithm more easily. In the FindSolution section, an adjustment option is provided to scale the points according to the user's existing input to meet the algorithm requirements.

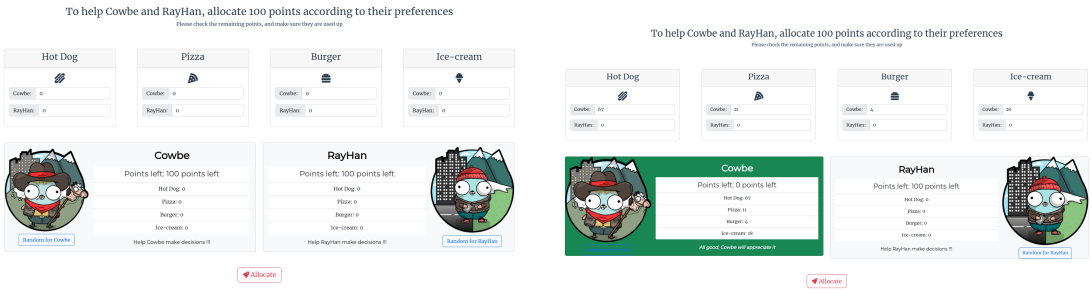


Fig. 6.9: Random option for visitors to experience the algorithm

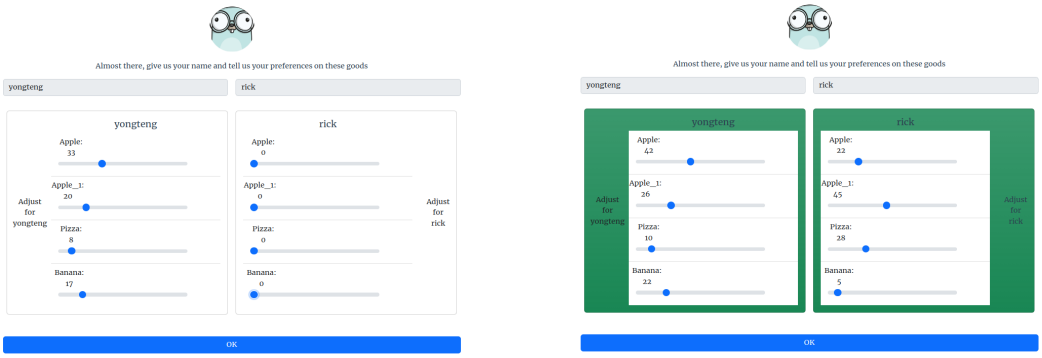


Fig. 6.10: Adjustment option makes input to satisfy requirements on existing scale

6.4 Terminal Application

The terminal applet is suitable for users who are looking for high efficiency, and who are in favor of *GRAB-AND-GO*.

(a) Start (b) Submit

(c) Allocate (d) Finish

Fig. 6.11: Get an allocation in under a minute

Chapter 7

Limitations and Future Work

7.1 Limitations

7.1.1 Algorithm

A key challenge for fair allocation is to address the different preferences and evaluations of individuals, as an allocation that is perceived as fair by one person may not be perceived as fair by another. Reaching consensus on what fairness means is also difficult because concepts like envy-freeness, proportionality, and efficiency may have different priorities. In addition, the complexity of finding the ideal solution increases with the number of participants and items involved, making it computationally challenging to achieve fairness in large-scale scenarios.

Althogh, the topic of this FYP only involves exploring the problem of fair allocation of items between two participants and the methods for a wide range of applications, but there are still some limitations.

- **Compromise between implementation and algorithmic ideas**

  There are multiple ways to implement algorithms. For example, Round Robin (RR) and Divide And Choose (DC) can be implemented very easily and efficiently because of the straightforwardness and practicality of the idea.

  However, algorithms like Adjusted Winner (AW) and Envy-freeness up to 1 (EF1) are more difficult to implement and do not have a common standard. In this implementation of FYP, the AW algorithm does not guarantee non-deprivability, which means that the item assigned to a participant may be deprived and participate in the next round of allocation, which is in violation of the original proposed Adjusted Winner.

As for the EF1 algorithm, since the algorithm attempts to guarantee PO by assigning the highest item preference to the participants at the beginning, this algorithm can be regarded as a greedy algorithm, which attempts to achieve the optimal solution but often falls short of the goal, i.e., PO, by local optimal solutions. Although it maintains some kind of fairness, a balanced number of items to be allocated, however, there are drawbacks in certain scenarios, refer to Tie input pattern in Shallow Test.

- **Differences between algorithmically perfect data and realistic data**

  Although realistic data are absent, it is possible to generate test sets and test the algorithm in a combinatorial way (possibly obtaining more coverage). However, because of the large number, and the absence of a realistic expected result against which to compare, the efficiency of the algorithm and the utility obtained can only be evaluated from the criterion mentioned in Section.4.2.1. But such a test lacks significance of the allocation to real-life scenarios.

7.1.2 User Interface

Although the current website has a good feel and look, with some robustness and a good interactive nature, there are still some limitations.

- The website has some lack of control over the box model, when dynamically changing prompts, sometimes the text will make the box bigger, causing some minor inconsistencies.
- Although the interface is responsive, this means that the website can be displayed properly on both PC and mobile. Likewise, the interface can sometimes look a bit "crowded" due to the lack of fine-grained component control.
- The site is still in the local testing phase, and is still one step away from being launched.

7.1.3 Engineering Management

In fact, it is quite thrilling to be an explorer in the field to do some experimentation. The problem of fair allocation is a niche, but relevant issue for everyone, something that has been studied since the Second World War, and has recently still been the subject of Nobel Prize winning research. Mainstream algorithm development is often done in Python programming languages, such as Fairpy. Many efficiency-conscious algorithms use C/C++ for more low-level control. However, Go, the most promising new programming language, has been less explored in this area. Fairalol, as an explorer of such attempts,

benefits from the inherent advantages of the Go language, and is about three times faster than Python implementation for the same behavior. However, Fairalol still has a major engineering drawback. For example.

- Only a simple project structure is managed. It is better to keep it simple when the initial project is not large enough, but it may create a potential problem for code bloat later.
- Absence of logical abstraction. The ideal approach would be to define an interface, implement the interface, and have the structure have its own methods, thus ensuring that the structure is bound by certain constraints to ensure that the logic is maintained. The current practice is for structures to call methods directly, which provides the convenience of pre-testing and modification, but is by no means a good specification.
- Code redundancy. The current test code has a lot of redundant code that is not properly extracted.
- Starter documentation. Beginner-friendly documentation and contribution guidelines should be made available for communication, and to motivate contributors.

7.2 Future Work

This FYP trip has been an exciting journey of exploration, not only in the field of fair allocation, a crucial issue in economics, game theory, and sociology, but also in the field of software development.

Regarding the algorithm, the basic principles and criteria for fair allocation have been somewhat justified. After that, a more global exploration of an allocation using linearly restricted computational methods can be explored, rather than greedy mathematical methods and mechanisms to achieve the goal.

Through this software development process, the front-end has been designed and implemented, and the back-end web service has been built from scratch, with the front- and back-end integration successfully completed. However, there is still room for improvement in refining the website interface, and the final step of deploying the website needs to be completed. Additionally, with the visitors' consent or proper desensitization of the data, relevant real-world data can be recorded and visitors can leave feedback on the allocation process. This feedback can then be incorporated into the algorithm to optimize the iterative allocation process.

This FYP project is not only an experiment and an innovation of the author, but also a pioneering and complete exploration process that can be referred to and improved by others. The source code of Fairallol's website is there, so readers can pull the code, build and access their own fairness algorithms to experiment, after the author refactored the code modularly and wrote detailed documentation.

The Fairallol backend code serves as an early explorer of fairness algorithms in Go, attempting to provide opportunities for like-minded peers to learn and improve, and the author will actively develop and improve the code and encourage potential contributors to contribute to the open source community and to the ongoing discussion of fairness algorithms.

Chapter 8

Conclusion

In this FYP, the author explores the problem of fair allocation of indivisible items in which two participants are involved. The four most widely used and effective methods in this setting, namely Divide And Choose (DC), Adjusted Winner (AW), Round Robin (RR), and Envy-freeness up to one (EF1), are explored and implemented as services in a web back-end written in Go for a front-end application written in the Vue framework. Through this FYP, the student experienced a complete journey of front-end web design and development, back-end deployment, and front- and back-end interaction. The four fair allocation algorithms explored were transferred to a modern, easy-to-use, robust, heuristic web interface, in addition to another terminal interface that could be accessed. But there are still some limitations here, as it is possible to collect real data on interface visitors and their comments under suitable conditions to feed the algorithm to optimize algorithms iteratively to achieve more realistic allocations. In addition, by using computational methods such as linear restrictions, a more global consideration of fair allocation can be carried out in later experiments to achieve realistic fairness (although the problem may be NP-hard, under certain settings, polynomial-level approximations can be found), instead of using mathematical methods and mechanisms to achieve the goal.

Thanks to FYP for providing the author with this valuable opportunity to explore the world and exercise his abilities. Fair allocation is a niche area of research and has a high threshold. It is almost impossible to find a clear tutorial on the web to start from scratch. This situation, however, motivates the author to read the literature, to study the mathematical expressions and the intrinsic meaning of the algorithms and to develop the ability to read pseudo-code and try to implement them. This provided a solid foundation for possible future research paths. In addition, this trip was really the fulfillment of a 4-year dream. To really build a front-end and back-end for research, and a complete interface open to the public, which is also relevant to my career plan. Thanks for the supervisor's

constant guidance and encouragement, the author so that have been able to do this.

Reference

[1] H. P. Young, *Equity: in theory and practice*. Princeton University Press, 1995.

[2] D. M. Kilgour and R. Vetschera, "Two-player fair division of indivisible items: Comparison of algorithms," *European Journal of Operational Research*, vol. 271, no. 2, pp. 620–631, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221718304764

[3] N. Bansal and M. Sviridenko, "The santa claus problem," in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, 2006, pp. 31–40.

[4] S. J. Brams and A. D. Taylor, "Fair division: From cake cutting to dispute resolution (j. oppenheimer)." *Public Choice*, vol. 93, no. 3/4, p. 514.

[5] S. J. Brams, D. M. Kilgour, and C. Klamler, "The undercut procedure: an algorithm for the envy-free division of indivisible items," *Social Choice and Welfare*, vol. 39, no. 2-3, pp. 615–631, 2012.

[6] S. J. Brams, P. H. Edelman, and P. C. Fishburn, "Fair division of indivisible items," *Theory and Decision*, vol. 55, pp. 147–180, 2003.

[7] S. J. Brams, D. M. Kilgour, and C. Klamler, "How to divide things fairly," *Mathematics Magazine*, vol. 88, no. 5, pp. 338–348, 2015.

[8] K. Pruhs and G. J. Woeginger, "Divorcing made easy." in *FUN*. Springer, 2012, pp. 305–314.

[9] D. Kurokawa, A. D. Procaccia, and J. Wang, "Fair enough: Guaranteeing approximate maximin shares." *Journal of the ACM*, vol. 65, no. 2, pp. 1 – 27.

[10] S. Bouveret and M. Lemaître, "Characterizing conflicts in fair division of indivisible goods using a scale of criteria." *Autonomous Agents and Multi-Agent Systems*, vol. 30, no. 2, pp. 259 – 290.

[11] I. Caragiannis, D. Kurokawa, H. Moulin, A. D. Procaccia, N. Shah, and J. Wang, "The unreasonable fairness of maximum nash welfare." *ACM Transactions on Economics and Computation (TEAC)*, vol. 7, no. 3, pp. 1 – 32.

[12] H. Aziz, B. Li, H. Moulin, and X. Wu, "Algorithmic fair allocation of indivisible items: A survey and new questions," *SIGecom Exch.*, vol. 20, no. 1, p. 24–40, nov 2022. [Online]. Available: https://doi.org/10.1145/3572885.3572887

[13] B. R. Chaudhury, T. Kavitha, K. Mehlhorn, and A. Sgouritsa, "A little charity guarantees almost envy-freeness."

[14] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi, "On approximately fair allocations of indivisible goods." New York, NY, USA: Association for Computing Machinery, 2004.

[15] E. Budish, "The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes." *Journal of Political Economy*, vol. 119, no. 6, pp. 1061 – 1103.

[16] B. Plaut and T. Roughgarden, "Almost envy-freeness with general valuations," *SIAM Journal on Discrete Mathematics*, vol. 34, no. 2, pp. 1039–1068, 2020.

[17] S. J. Brams and A. D. Taylor, *The win–win solution: Guaranteeing fair shares to everybody*. WW Norton & Company, 2000.

[18] X. Bei, X. Lu, P. Manurangsi, and W. Suksompong, "The price of fairness for indivisible goods," *Theory of Computing Systems*, vol. 65, pp. 1069–1093, 2021.

[19] H. Aziz, H. Moulin, and F. Sandomirskiy, "A polynomial-time algorithm for computing a pareto optimal and almost proportional allocation," *Operations Research Letters*, vol. 48, no. 5, pp. 573–578, 2020.

[20] G. Amanatidis, G. Birmpas, and E. Markakis, "On truthful mechanisms for maximin share allocations," *arXiv preprint arXiv:1605.04026*, 2016.

[21] H. Aziz, B. Li, and X. Wu, "Approximate and strategyproof maximin share allocation of chores with ordinal preferences," *Mathematical Programming*, pp. 1–27, 2022.

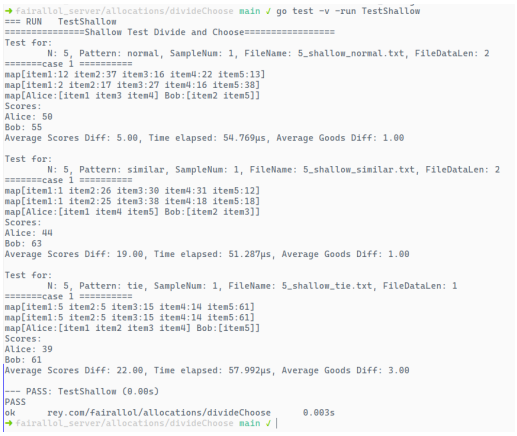
[22] H. Varian, "Equity, envy and efficiency," *J. Econ. Theor.*, vol. 9, p. 63–91, 1974.

[23] L. E. Dubins and E. H. Spanier, "How to cut a cake fairly." *American Mathematical Monthly*, vol. 68, p. 1.

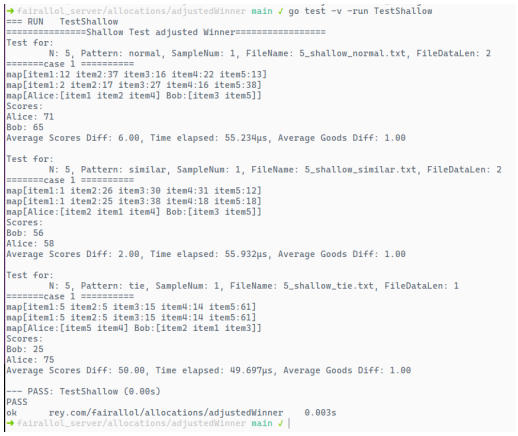
[24] A. Procaccia and J. Wang, “A lower bound for equitable cake cutting.” in *EC 2017 - Proceedings of the 2017 ACM Conference on Economics and Computation*, no. EC 2017 - Proceedings of the 2017 ACM Conference on Economics and Computation, pp. 479–496 – 496.

Appendix A

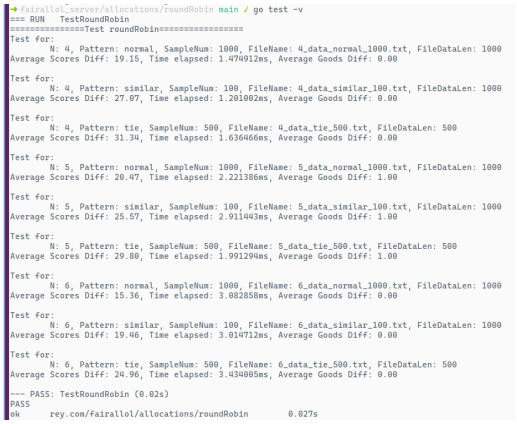
Shallow Test Evidences



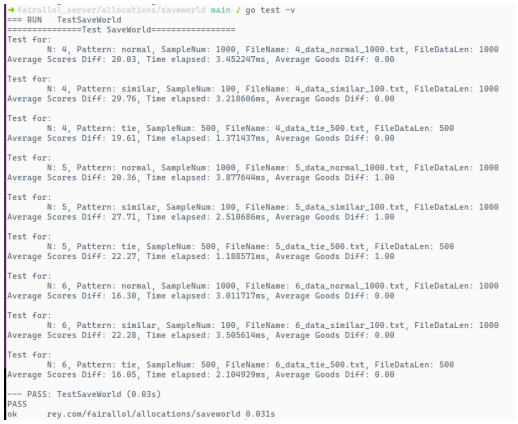
(a) Shallow Test for DC



(b) Shallow Test for AW



(c) Shallow Test for RR



(d) Shallow Test for SW

Fig. A.1: Shallow Test

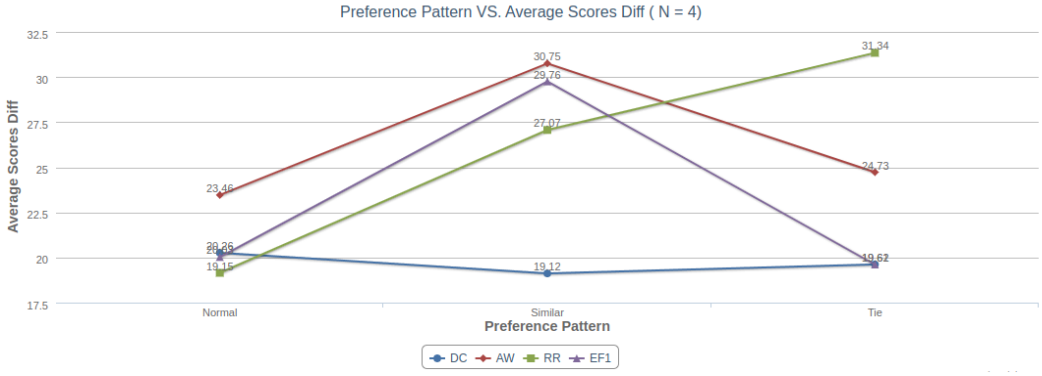
```
Test for:
        N: 4, Pattern: normal, Time elapsed: 0.03606128692626953
Test for:
        N: 5, Pattern: normal, Time elapsed: 0.07582902908325195
Test for:
        N: 6, Pattern: normal, Time elapsed: 0.11426568031311035
Test for:
        N: 4, Pattern: similar, Time elapsed: 0.034507989883422285
Test for:
        N: 5, Pattern: similar, Time elapsed: 0.07597112655639648
Test for:
        N: 6, Pattern: similar, Time elapsed: 0.112120389938354449
Test for:
        N: 4, Pattern: tie, Time elapsed: 0.031468152999877793
Test for:
        N: 5, Pattern: tie, Time elapsed: 0.06951165199279785
Test for:
        N: 6, Pattern: tie, Time elapsed: 0.11029601097106934

[Process exited 0]
```

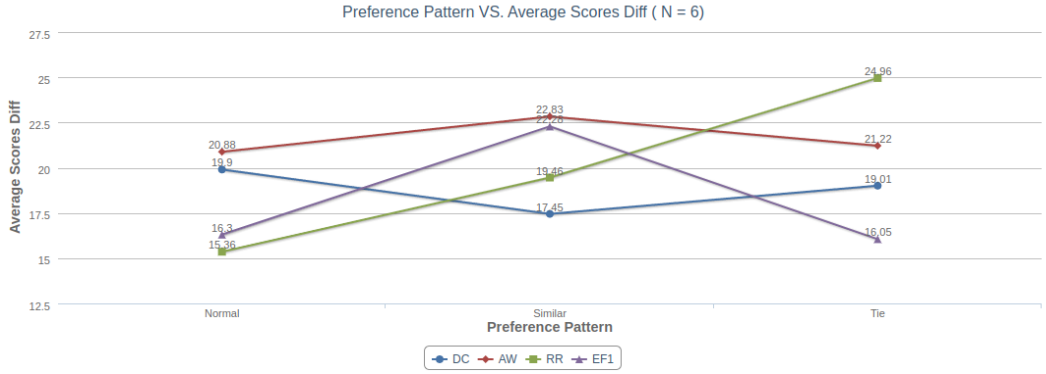
Fig. A.2: Shallow test for Fairpy (The Same behavior as Fairallol)

Appendix B

Overall Analysis

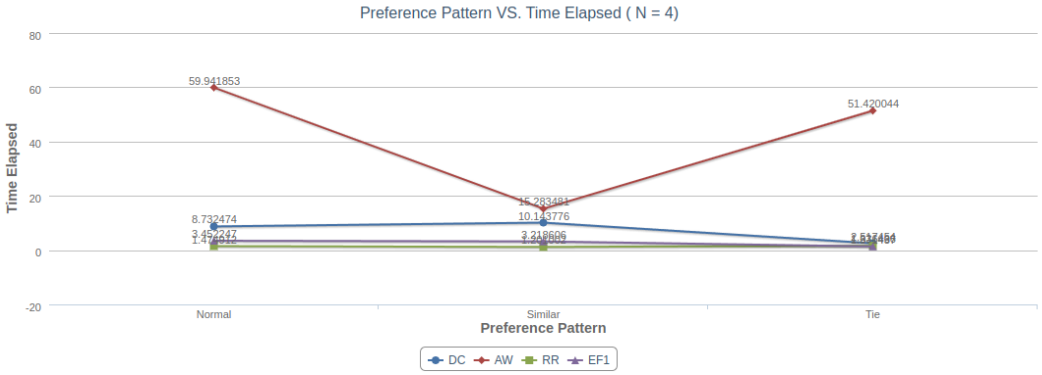


(a) Preference Pattern VS. Average Scores Diff (N = 4)

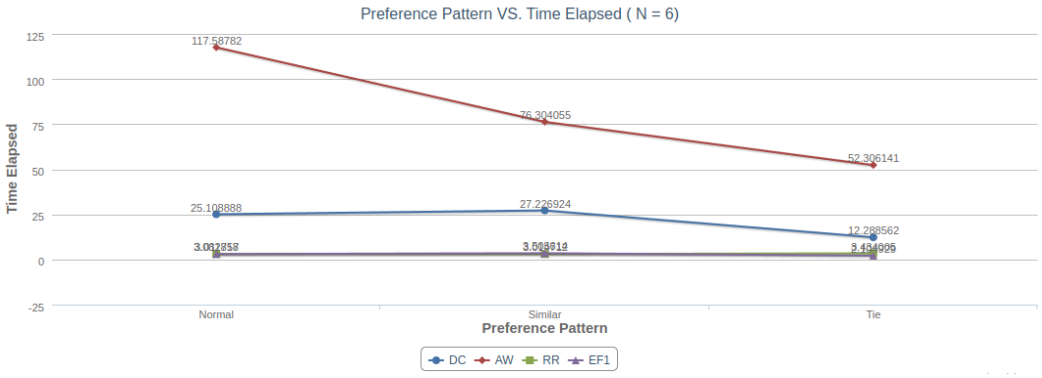


(b) Preference Pattern VS. Average Scores Diff (N = 6)

Fig. B.1: Overall Analysis - Pattern VS. Average Scores Diff



(a) Preference Pattern VS. Time Elapsed (N = 4)

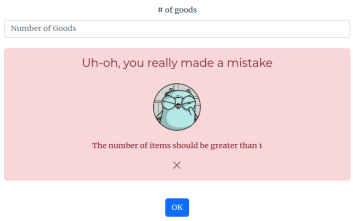


(b) Preference Pattern VS. Time Elapsed (N = 6)

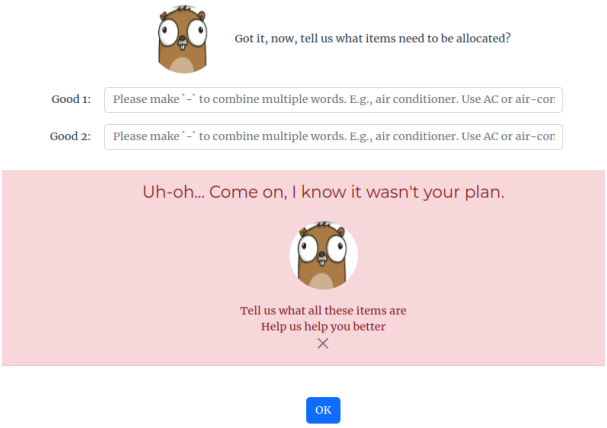
Fig. B.2: Overall Analysis - Pattern VS. Time Elapsed

Appendix C

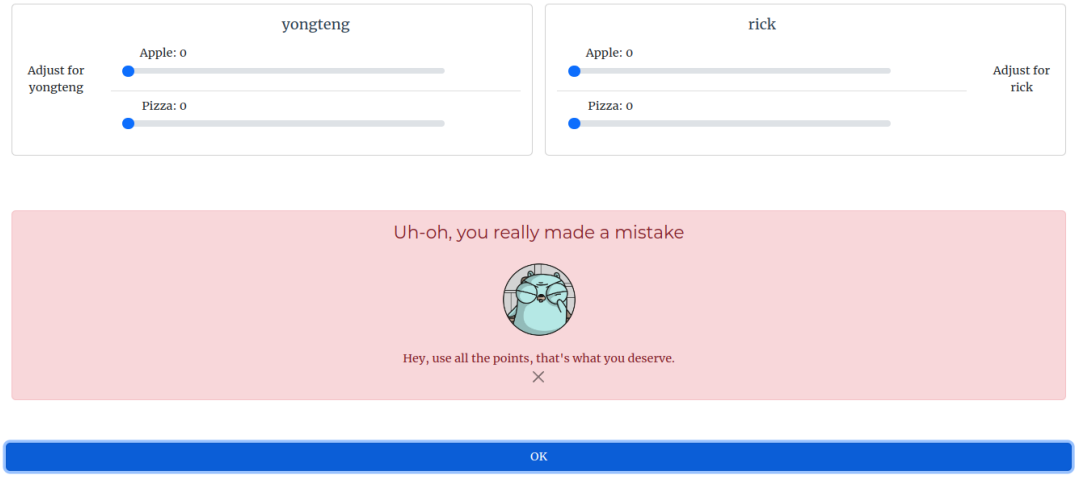
Hints at Find Solution Page



(a) Input illegal or all points are not used up

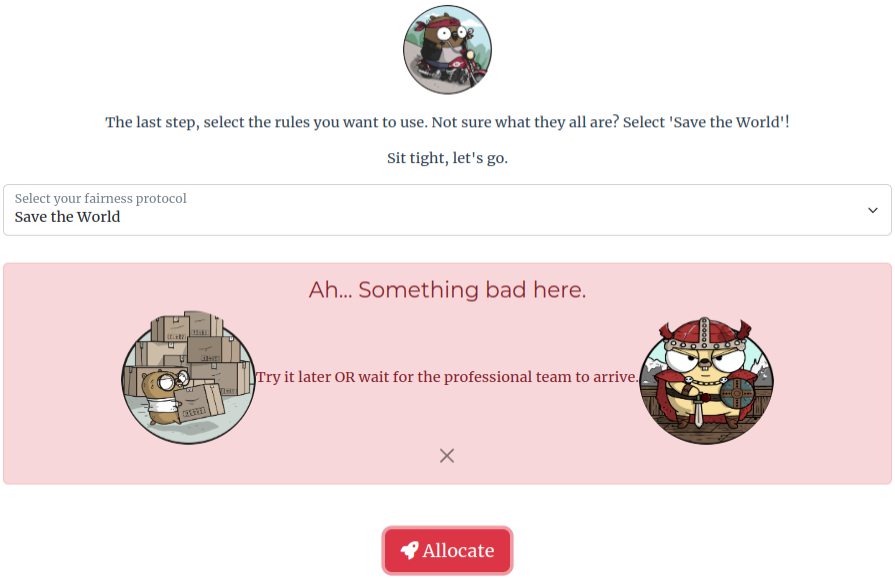


(b) Network errors or back-end crashes

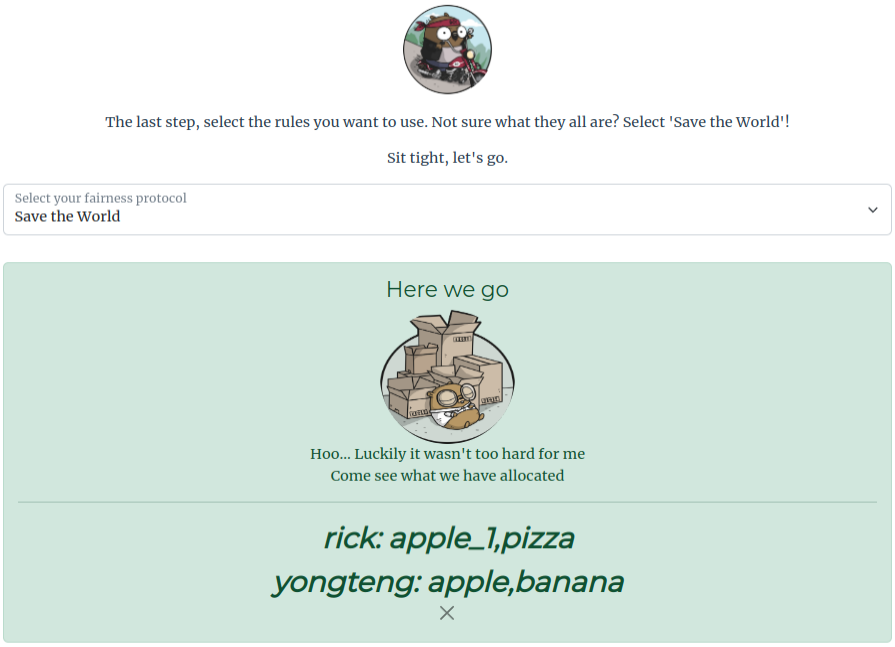


(c) Everything is going well

Fig. C.1: Hints at FindSolution1



(a) Network errors or back-end crashes



(b) Everything is going well

Fig. C.2: Hints at FindSolution2

Appendix D

Website Robustness Example

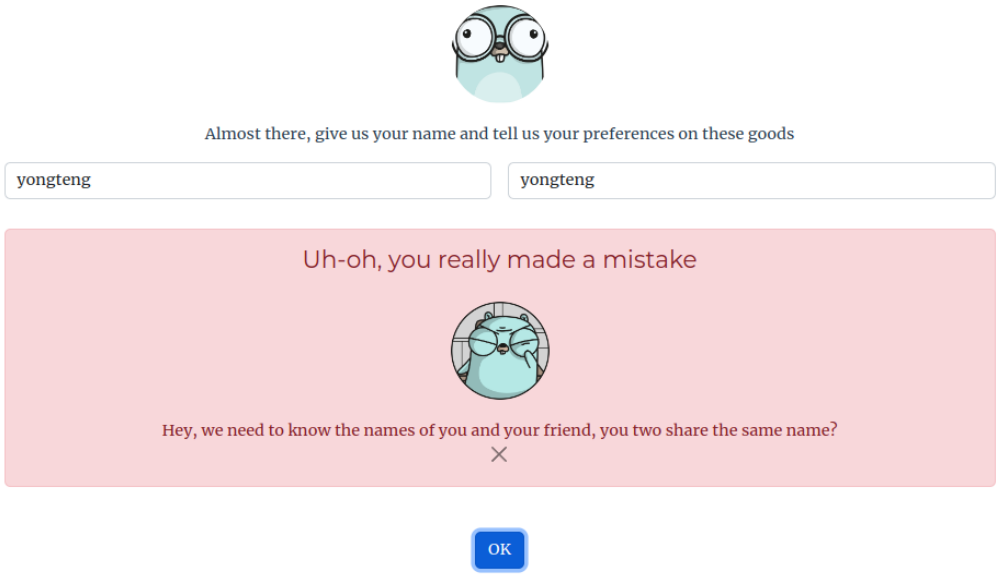


Fig. D.1: Front-end application intercepts duplicate names