Xi'an Jiaotong-Liverpool University

西交利物浦大学

DTS311TC 毕业设计

研究公平分配算法并通过易于使用的界面使其得到广泛应用

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

By

雷永腾

Yongteng Lei

ID: 1930236

Supervisor Name

Dr. Md Maruf Hasan

School of AI and Advanced Computing

Xi'an Jiaotong-Liverpool University

April 2023

摘要

本次的 FYP 将致力于研究涉及两个参与者或代理（Agents）持有不同偏好的不可分割物品的公平分配算法。事实上，该项目将公平分配问题视为社会选择问题，更侧重于现实生活场景而非博弈论。换句话说，该项目将偏好信息视为两个代理人在给定情景中的真实想法，而不考虑隐藏或歪曲事实的个人原因。然后，这些算法会被应用到一个易于使用的界面上，这里指的是一个网页，可以作为解决日常实际问题的参考，如遗产的继承、多余物品的分配、任务分配等。

关键词: 公平分配算法；算法实现；网页开发；用户界面；Golang；

目录

图片目录

表格目录

术语表

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

引言

公平是人们在理论和实践中一直追求的目标 [1]。人们希望根据参与者的偏好，创造出既能满足效率标准又能满足公平标准的分配方案，这推动了这一领域算法的发展。随着时代的变迁，数学、经济学、政治学、计算机科学等领域都从不同的角度和目的对公平分配理论进行了研究。公平分配资源在集体选择环境中占据重要地位也就不足为奇了。

公平地分配无限可分的资源已经十分困难，但当资源由有限的不可分物品集合组成时，事情就变得更加复杂 [2]。每个参与者总是会根据他们对这些物品的评价收到被称为 “Bundle” 的资源子集。我们必须根据参与者对这些物品的偏好进行分配，同时考虑到一个或多个公平和效率标准。考虑 “两个小朋友分配三个巧克力棒” [1] 的情况，如果没有合理的公平标准，也没有任何一个参与者妥协，分配结果将永远是一个死局（Tie）（总是有一个代理人拥有更多的糖果）。为了证明公平分配物品是多么具有挑战性，我们简化设定，假设参与者的效用（Utility）是线性的，并且没有协同效应 [2]，并考虑 “圣诞老人问题”，即如何将有限数量的礼物分配给有限数量的孩子，同时没有不开心的孩子，并让效用最大化？这就是圣诞老人应该做的！但即使只考虑一个公平标准，这个问题也是一个 $NP\text{-}hard$ 的问题。而且人们相信，不存在近似的有效解决方案 [3]。

无法再分割的物品的分配是一个不容忽视的重要研究课题。这一方面是因为在这种情况下可能不存在 “公平” 的分配，另一方面是因为一些直接的算法会因为物品不可分割的特性而失败，而这在现实生活中是经常发生的。要做到这一点，一个简单的方法是让参与者排成一排，按偏好顺序循环给予他们当前最想要的物品。这听起来很合理，也很有说服力，但如何让它们排队却是个问题。幸运的是，许多有才华的作者最近提出了各种针对具体问题的分配算法，这些算法都能满足一

[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).

定的公平性和效率标准。

本 FYP 将研究公平分配的算法和概念，同时考虑实现的可能性，并将其转化用于易于使用的界面，以供日常分配问题参考 [4] [5] [6] [7] [8]。

Chapter 2

前置知识

本研究将重点关注不可分割商品的公平分配问题。考虑一个公平分配问题实体 $E = (v, M, N)$ 。N 是 n 代理人的集合。$v = (v_1, v_2, v_3, \cdots, v_n)$ 是一个非负的附加值向量（additive valuations vector），代理 $i \in N$，注意 $S \subseteq M$。并且 $v_i(S) = \sum_{j \in S} v_i(j)$，其中，$v_i(j)$ 表示代理人 i 对 M 中的商品 j 进行估值的集合，S 是将物品被分成 n 个包 (Bundle) 的子集。我们的工作是在这个实体 E 中，根据相应的公平标准，尝试将 $|M|$ 个物品公平地分配给 $|N|$ 个代理，这些标准通常被称为 *MMS*、*PFS* 和 *envy*。和 *envy-free* 等。

Definition 1 (Maximin share guarantee)**.** MMS 保证代理 $i \in N$ 满足

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

这意味着代理人将把物品分成 n 包，他总能被保证至少得到最想要但最效用最小的一包。值得注意的是，MMS 分配（保证每个代理人都能分到自己的 MMS）并不总是存在的。但幸运的是，Procaccia 和 Wang [9] 证明了至少（$\frac{2}{3} - MMS$）近似的 MMS 分配 (approximately MMS allocation)（每个代理人将得到 $\frac{2}{3}$ 他/她的 MMS）总是存在的。

Definition 2 (Proportional fair-share)**.** PFS 分配保证代理 $i \in N$ 得到在整个物品集合 M 的 $\frac{1}{n}$ 的效用，也就是说 $PFS = \frac{v_i(M)}{|N|}$. 一个 proportional allocation 是对于每一个代理在集合 N, 有:

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

每一个代理将得到他/她的 PFS (total value / n)。值得注意的是 *PFS allocation* 意味着满足 *MMS*, 如果每一个代理都有着可加性的效用 (superadditive utility) [10]。

Definition 3 (Envy-freeness)**.** EF 意味着所有的代理相比其他代理分配的物品都更喜欢自己的。有

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

对于两个代理 i 和 j, 有 $\alpha \in (0, 1]$

当 $\alpha = 1$, 我们将它称为 *EF*.

Definition 4 (Envy-freeness-except-1)**.** 当 agent$i$ 嫉妒 j, 但是当 j 的最有价值的 (*the most valuable*) 物品被去除后，嫉妒消失，那么就满足 EF1，有

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

对于两个代理 i 和 j, 当 $\alpha \in (0, 1]$，符号 "\" 代表着从包 (Bundle) S_j 中去除一个物品 g。

当 $\alpha = 1$, 我们将它称为 *EF1*. EF1 比 EF 更加宽松，并且 Maximum Nash Welfare 算法同时满足 EF1 和 Pareto-efficient [11]。

Definition 5 (envy-free up to at most any item)**.** 当 agent$i$ 嫉妒 j, 但是当 j 的最低价值的 (*the least valuable*) 物品被去除后，嫉妒消失，那么就满足 EFx，

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

对于两个代理 i 和 j, 有 $\alpha \in (0, 1]$

当 $\alpha = 1$, 这个公平准则被称为 *EFx*. 尽管 EFx 比 EF 宽松，但是比 EF1 严格的多。对于公平而言，EFx 比 EF1 更加合理，但也更加难以实现。事实上，EFx 是否广泛存在仍然是一个开放问题 [12].

将实体 E 分配为 $v = (\emptyset, \emptyset, \cdots, \emptyset)$ 仍然满足 EF、EFx 和 EF1。然而，让所有项目 M 都捐赠给慈善机构对代理来说没有任何意义，尽管没有人会羡慕慈善机构集 [13]。显然，我们还需要一些建立在纯粹无嫉妒（pure envy-free）基础上的效率指标。

Definition 6 (Pareto efficient)**.** 帕累托效率（Pareto efficiency）或帕累托最优（Pareto optimality）（PO）是这样一种情况：对于 N 中的所有 i，不存在替代分配 v'，使得 $v_i(S_{i'}) \geq v_i(S_i)$。

换句话说，不存在代理人 $i \in N$ 可以在不影响他人价值的情况下提高自身价值的 PO 分配。

我们将始终牢记这些基本原理，并将其作为探索和实施公平分配算法的基础。

Chapter 3

文献综述

如前所述，FYP 主要关注的是不可分割货物的公平分配（货物将被视为一个整体，不能再被分割成多个部分），此外，本节还提到了一些可用于公平分配的网站。

3.1 不可分割物品的公平分配

当谈及分配的物品不可再被分割时，就无法保证精确的无嫉妒和公平概念。因此，科学家们提出了更宽松的公平（fair relaxations），如 EF1、EFx、MMS 等，它们定义了“一定的”公平 [11] [14] [15] 。许多计算机科学家已经提出了很多有效的通用算法，这里回顾一些最常用且强大的算法，它们也是更复杂环境下算法的基础。

3.1.1 Divide-and-Choose (DC)

Divide-and-Choose 是最经典的分配算法之一，在只有两个代理人的情况下，这个过程直观而高效。第一个代理人有权把物品分成几组，由第二个代理人来选择，因此最有效、最公平的分法是**第一个代理人尽可能使每组的价值最大化**，以确保自己也能获得最大利益。如果第一个代理人最大化了 (X_1, X_2) 中较小的组，那么 MMS 对他/她总是成立的，而 EF 对第二个代理人是有效的，因为他或她总能从 (X_1, X_2) 中得到了最喜欢的组。这一点已由 Plaut 和 Roughgarden 证明 [16]。

3.1.2 Adjusted-Winner (AW)

另一种广泛应用于两个代理场景的分配算法是 Adjusted-Winner。该算法会根据已经获得的效用来动态调整两个代理对物品的评估 [17]。对于第一个被分配物品的代理时，EF1 始终存在。这种算法保证了两个代理之间尽可能高的社会福利，但并不一定满足 MMS 或 EFx [18]。

3.1.3 Sequential Allocation and Round-Robin (RR)

另一类可以高效实现的公平分配算法是顺序分配法，也称为顺序挑选法，即人们按照一定的顺序挑选自己喜欢的物品，而这个顺序一直有效 [17]。Round-Robin algorithm 是最流行的顺序算法之一，它对不可分割的物品和家务 (chores)[1] 重复一个顺序模式 1...n，以保证 EF1，但 EFx 可能并不总是存在。Aziz 等人 [19] 针对不可分割项目和杂务的混合问题提出了一种双循环解决方案。Amanatidis 等人 [20] 和 Aziz [21] 等人通过顺序算法近似了 MMS 公平性。

3.1.4 Envy-freeness (EF)

无嫉妒（EF）是一种理想化的公平标准，它要求每个参与者在分配物品时都不会嫉妒他人的分配。即使参与者和产品数量很少，要达到这种公平程度也是非常困难的 [4][22]。这一领域的早期贡献包括 Stromquist [23] 对住房分配问题的研究，以及 Varian [24] 对经济学中公平分配的研究。

针对 EF 的局限性，研究人员提出了更实用的公平标准，如 EF1 和 EFx。EF1 要求每个参与者的项目分配不能让他们过分羡慕其他参与者的分配 [11]。这一要求很容易满足，而且在很多情况下被认为是足够公平的。在不可分割的项目分配中，Plaut 和 Roughgarden [16] 提出了一种多项式时间 (polynomial-time) 技术来实现 EF1。考虑到比 EF1 更强但并非 EF 难以实现的标准 EFx，要求参与者不能通过删除任何一项来羡慕他人的分配，然而，EFx 的广泛可用性仍是一个悬而未决的问题 [12]。

3.2 相关工作

3.2.1 Spliddit

Caragiannis 等人 [11] 提出了一种巧妙的方法，将可分割问题的相关算法 MNWs（最大纳什福利）转换为不可分割问题，他们证明了 MNWs 和 MMS 一样，在不可分割的情况下仍然意味着一定的 EF1。最后，他们发现，在这种情况下，MNW 是一个 *APX-hard*，一个 *NP* 优化问题，为此，他们设计了一个特殊的技巧，给每个代理 1000 点，并使用多项式时间规约 [2] 来克服这一困难。最后，他们用这种算法作为 Spliddit 网站的底层实现 (Fig. 3.1)。

[1] But not a mixture of them

[2] Polynomial-time reduction.

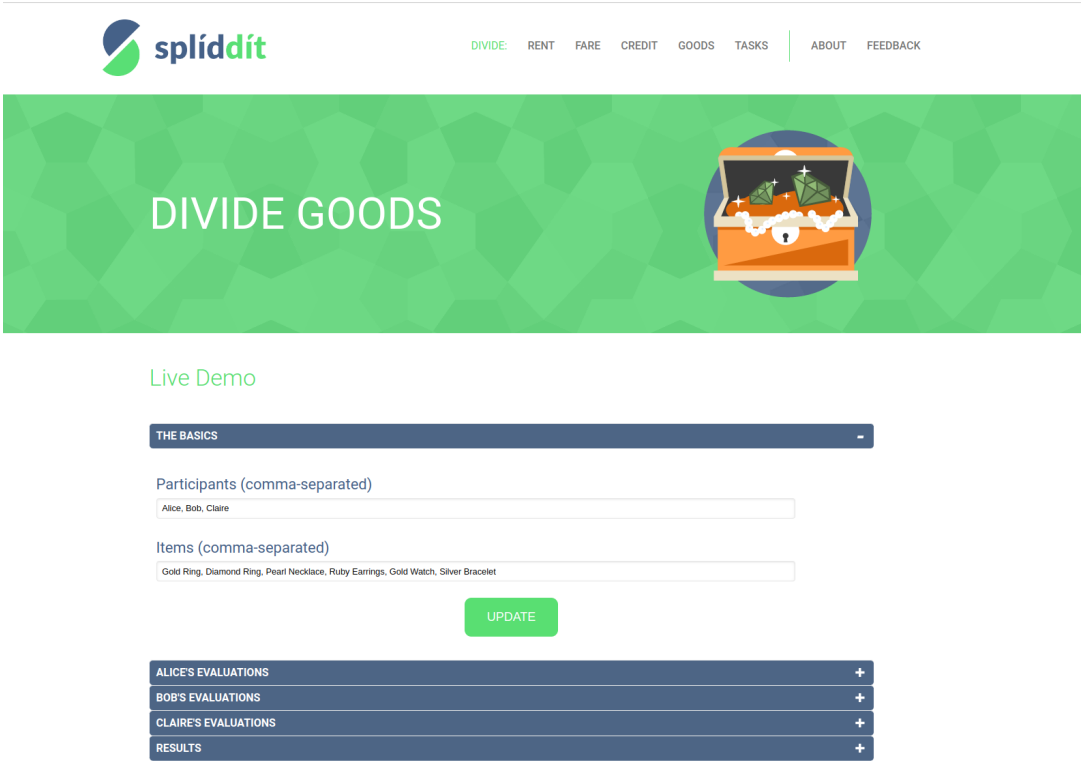
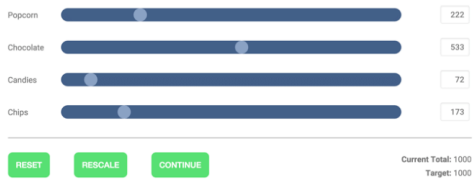
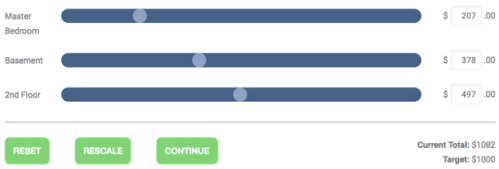


Fig. 3.1: Spliddit

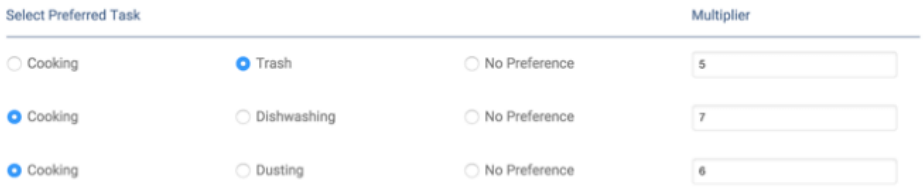
Spliddit 的设计旨在解决三个具体的公平分配问题：分摊租金、分配物品和分享信用。每个问题类型都有详细的解释、教育文章链接以及演示。但它目前已停止服务。



(a) Spliddit-goods-input-interface



(b) Spliddit-rent-input-interface



(c) Spliddit-tasks-input-interface

Fig. 3.2: Spliddit-input-interfaces

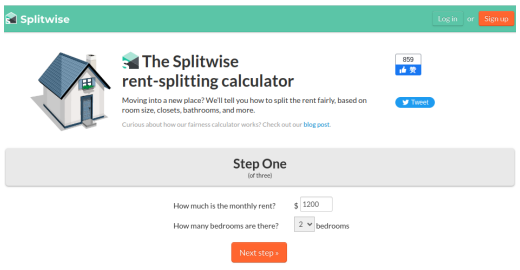
在物品分配任务中，参与者每人有 1000 个点来表示他们对给定商品的偏好。无论

算法是基于序数效用（ordinal utility）[3] 还是基本效用（cardinal utility）[4] 都可以以此作为直接输入或间接转化来满足算法的不同输入需求。同样的情况也发生在租金分配中，不同的是，总点数就是租金总额，参与者根据自己的偏好对不同的设施给出不同的报价（offer）。

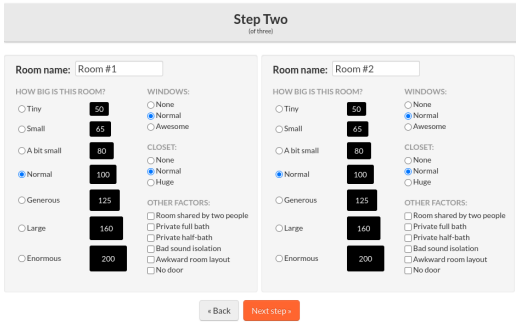
在 Spliddit 任务分配中，多个用户需要合作完成一项任务，每个用户提供一个输入信号（input signal，例如一个偏好的提交），用于计算任务结果。当多个用户都提交输入信号时，这些信号将通过多路复用器（multiplexer）进行编码和选择，从而产生一个可传递给任务算法进行计算的组合信号（combined signal）。

3.2.2 Splitwise

Splitwise 房租计算器也是一个基于网络的工具，它简化了室友之间的房租分配，促进了合租情况下的公平性，它基于诺贝尔经济学奖得主 Alvin Roth 的公平原则，用户需要提交总房租、室友人数和房间属性。然后通过迭代过程分配租金，修改每个房间的租金，直到各方就分配达成一致。

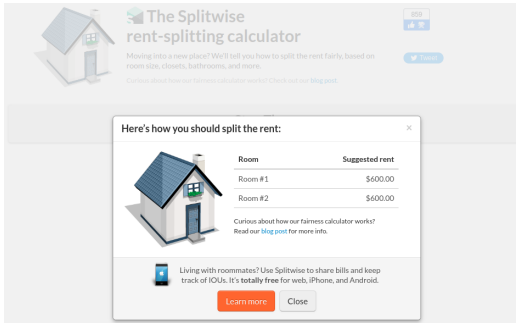


(a) Splitwise step1



(b) Splitwise step2

(c) Splitwise step3



(d) Splitwise result

Fig. 3.3: Splitwise

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

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

Spliddit 通过分配点数来完成对参与者对商品偏好的调查，提供了极大的可扩展性，以适应基于序数效用和心数效用的算法。Splitwise 房租计算器简单而美观的界面也是网页设计的灵感来源。

3.2.3 Adjusted Winner Website - NYU

另一个由纽约大学维护的网站（Fig. 3.4）展示了由 Steven J. Brams 和 Alan D. Dedicated 提出的 Adjusted Winner，AW 算法的过程，以尽可能公平地在两个参与者之间分配 n 个可分割项 [4]）。该网站还提供了一些关于公平分配数学的背景信息，以及自定义数据的场所（playground）。虽然该网站的重点是可分割物品，但其教学方法与本次 FYP 目的预期成果一致，既为公平分配任务提供了参考，又让人们对算法如何实现公平有了一定的了解。

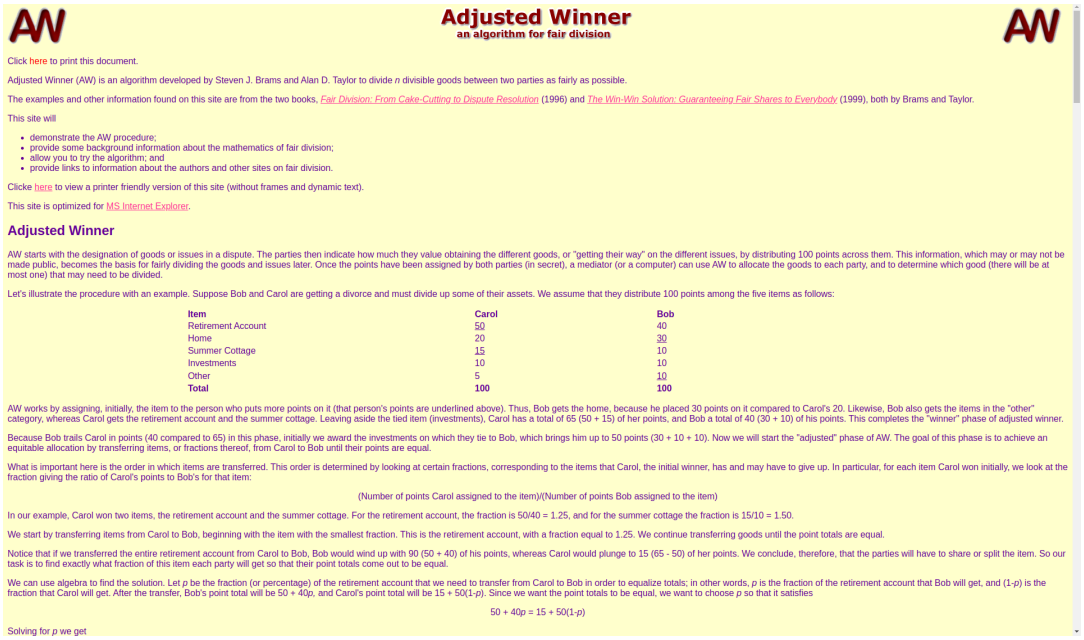


Fig. 3.4: A website showcasing Adjust Winner algorithm maintained by NYU

3.2.4 Fairpy

Fairpy [5] 是一个基于 Python 的公平分配算法库，旨在为分配问题提供各种公平的解决方案。该库涵盖了公平分配的几个重要概念，包括 envy-free、propotional 和 fairness。它为不同的情况提供了多种算法，例如可分割和不可分割的物品，以及有无货币转移的情况。

Fairpy 的设计对初学者十分友好，提供清晰的文档和示例，以方便实施与部

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

署。研究人员、经济学家和开发人员可以使用该库来探索、分析和解决各种应用中与公平相关的问题，如资源分配、日程安排和协作决策等。

通过为公平分配提供一套全面的工具和资源，Fairpy 为学术界和实践界围绕公平和公正的持续讨论做出了贡献。

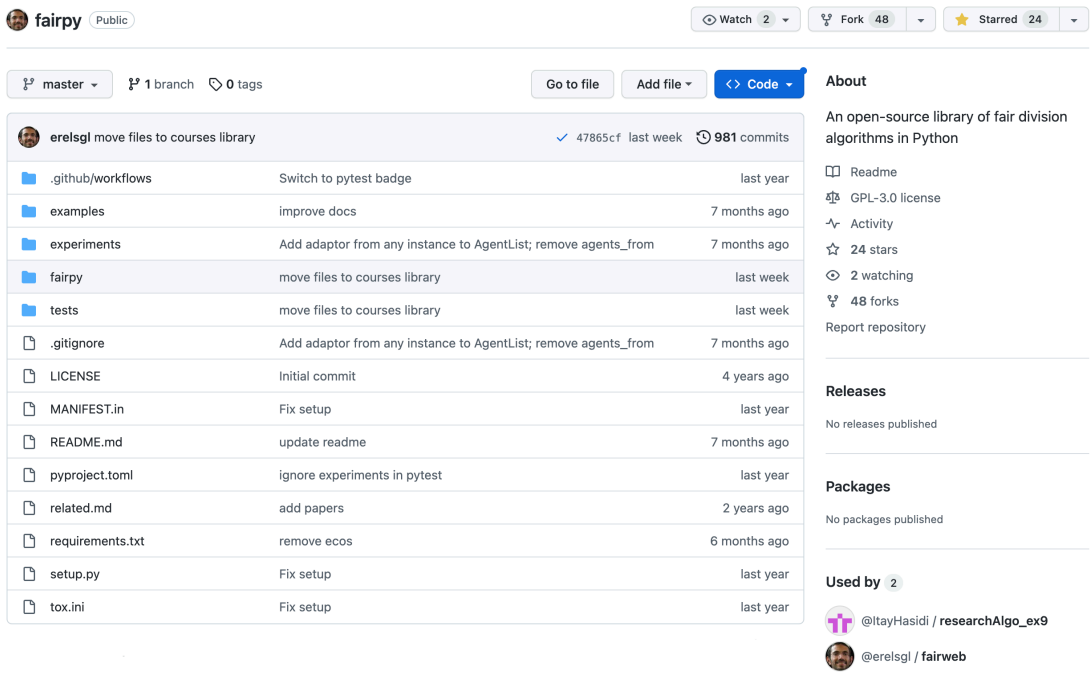


Fig. 3.5: Fairpy – a comprehensive set of tools and resources for fair division

得益于这样的用户友好网站、强大的算法和经济原则基础，它们在共享资源和财务管理领域具有重要的参考价值。本 FYP 将重点探索上述四种广泛使用的理念和方法（DC、AW、RR、EF），通过创新、易用、启发式的网站服务，唤醒公众的公平分配意识，为生活提供一些现实的便利。

Chapter 4

实验方法

4.1 探索与反思（理论方法 Theoretical Method）

公平分配是一个跨领域的话题，涉及经济学、博弈论、数学和计算机科学等多个学科。每个领域都要应对独特的挑战，并以不同的方式解决不同环境下的问题。

为了深入了解问题，全面的文献综述以及证明和应用场景研究对于验证算法的工作过程至关重要。此外，探索网络上的现有界面，如经过高度验证的流行博客和网站，也能为简单易懂的解释性内容和用户友好的功能提供有价值的见解。

4.2 设计与实现（实践方法 Empirical Method）

本 FYP 包括两个主要部分，即算法探索和接口实现。算法部分侧重于探索四种广泛使用的算法（DC、AW、RR、EF），并尝试使用 Go 语言实现和测试相应的分配行为（迄今为止，Go 语言中还没有公平分配的先驱）。

在界面实现部分，有两种可访问的方式，即基于网络和终端小程序。基于网络的界面旨在创建一个创新、醒目、易用、教育性强的公平分配解决方案，具有强大的指导性和强大的功能。终端小程序的重点是可访问性（根据终端的特性，不是每个人都愿意使用，但它却是随时可用的，甚至是更加高效的解决方案）。

4.2.1 第 1 部分：算法实现与测试

算法实现与创新

所选的四种算法都将使用 Go 语言实现，其中 DC 和 RR 是经典实现的复现。对于 AW 和 EF 算法，我们将原有理论作为参考，并以此为基础实现两种更现实的算法，尽管它们在某些严格的公平性上做了某种妥协。

算法评估标准

在深入查阅文献后，应挑选几种适合在本科阶段探索和实施的算法，并从几个方面对其性能进行测试。

- **完备性（Completeness）**：算法能否将所有物品分配给所有参与者。
- **公平性（Fairness）** [1]: 参与者在分配后获得的效用差。
- **效率（Efficiency）**：运行时间，以及完成分配所需的步骤数。

算法输入模式

不同的算法可能会对其输入很挑剔。测试将考虑 3 种不同的输入模式（patterns）的性能，以确保算法的实现具有一定的通用适应性。

- **普通（Normal）**所有参与者对所有待分配都有一个正常 [2] 的评估，而不存在刻意的相似的偏好。
- **相似（Similar）**所有参与者对待分配物品具有大致相同的偏好。
- **僵局（Tie）**所有参与者对待分配物品具有完全相同的偏好。

测试数据准备

大多数情况下，要测试算法在公平分配案例中的表现，真实世界的数据十分重要，如多余公共物品的分配、离婚资产分割、家务分配等场景的真实数据。不过，在没有这些数据的情况下，生成模拟数据也是一种有效的替代方法 [3]，有助于评估算法的有效性。可以准备以下测试数据：

- 使用 Python 程序生成一个由 N 个数字组成的穷举数列，使其总和为 100（对标 Spliddit 的 1000 点）。选择 4、5 和 6 的 N 值（生成 $N = 7$ 的数据需要超过 32G 的内存支持）。从数列中随机选择 1000 个数据项，并将两组数据作为一对作为算法的数据集进行测试。这样就产生了 500 个测试用例，模拟正常输入模式。

[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.

[3] Real-world data is always a combination of a finite number of natural numbers, at least, in this case.

- 重复上述过程，但确保每 10 个数据项都有相似的偏好。这将创建 500 个测试用例，但其中包含 100 个相似模式，作为算法的数据集。模拟相似输入模式。
- 随机选择 500 条数据，并将每个数据共享给两名参与者。这样就产生了 500 个的测试用例，每个参与者对物品的偏好相同。模拟僵局输入模式。

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

通过模拟这些场景，可以测试算法在不同情况下的有效性，并评估其性能。

总体分析

使用 Fairpy 进行外部测试分析 在完成所有算法的实施和测试后，将对 Fairpy 和 Fairallol [4] 进行外部测试，以验证两个主要方面。

- 相同的算法是否会产生一致的行为。
- 效率比较（运行时间）。

内部测试分析 此外，还将在 Fairallol 中使用第 4.2.1 节定义的方面进行效率比较分析，以评估所实施算法的优缺点，并得出总体总结。

[4]The name of this project.

4.2.2 第 2 部分：界面设计与实现

网页界面设计

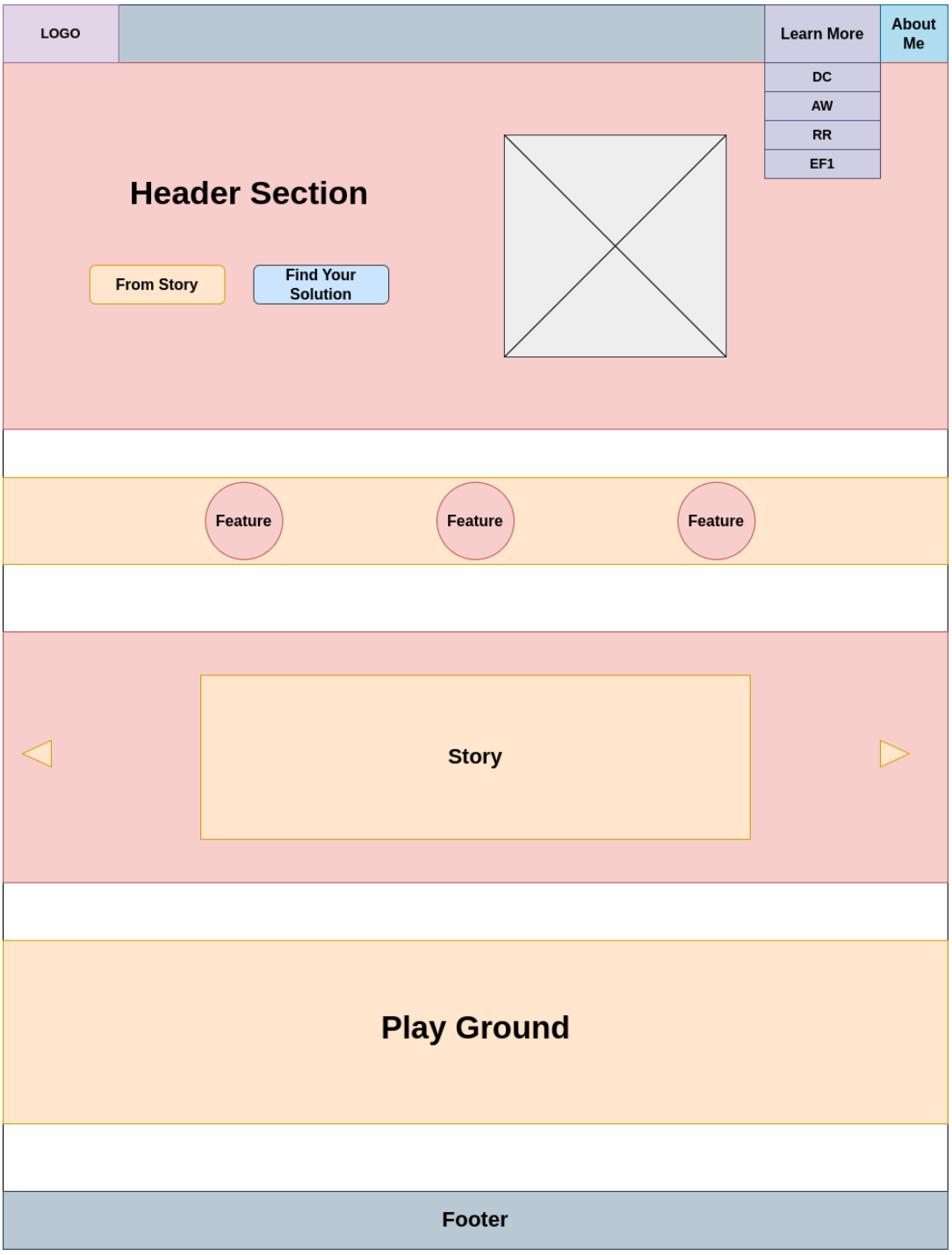


Fig. 4.1: Wire frame of Fairallol

Fairallol 网站使用的技术

网站名称为 Fairallol[5]，使用 Vue 框架构建，采用 HTML、CSS、JavaScript 等技术。网站使用 Bootstrap5 作为组件库，NerdFonts Icon 则提供了一些可爱的图标。Axios 发送请求与后端服务器通信，接收响应并处理错误。

后端服务使用 Go 语言的网络框架 *Gin* 编写，两个复现的经典实现（DC 和 RR）以及基于 AW 和 EF 的两种算法被用作服务。

所有前端和后端代码均由作者独立编写。[6] [7]

网站内容设计

Fairallol 是新一代创新型公平分配网站，其界面具有互动性、参与性和教育性。网站试图通过生动的实例和与参与者的互动，唤醒人们对公平分配问题的思考和理解，并为他们的生活提供切实的便利。

因此，Fairallol 的内容经过精心设计、易于阅读、引人入胜、富有启发性，每种算法的解释都通俗易懂，便于初学者理解，并有相应的 playground 供访问者自由探索。同时，功能强大，且用户体验友好。

评估

- 检查算法是否包含完整的证明和必要的解释。
- 对该算法进行实现分析，并评估其实用性。
- 分析网站的可用性和操作。(是否 bug-free 且易于使用)

预发布测试（Minor Testing）

在 Fairallol 发布之前，应进行一次小型的发布前测试，并将网站分发给一小部分人，以测试用户界面等的健壮性、功能性和易用性。

[5]Fair Allocation LOL

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

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

4.3 理论方法（Theoretical）与实践（Empirical）方法之间的联系

实现不可分割物品的公平分配过程充满挑战。全面的文献综述至关重要，因为这关系到可以挑选适当的算法从而专注于实现，而不必过于关注严格的证明或深入研究主要的应用场景。通过对多种公平原则的深入研究，我们可以正确理解它们之间的相互关系，例如：EF、EFx、EF1 和 Prop1（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

这种对公平原则定理的全面调研不仅能减少总体工作量，还能深入了解每种已实现算法所符合的具体标准。因此，研究人员可以更有效地专注于改进算法和评估其有效性。通过文献研究加深对公平性标准的理解，可作为未来算法开发的基础，并确保公平性标准在所采用的方法中得到适当体现。

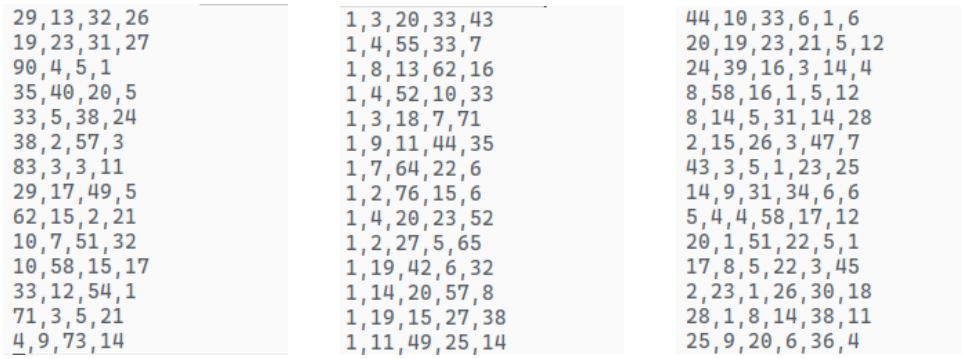
Chapter 5

实验

5.1 数据准备

如前所述，由于没有真实数据，因此将使用完整的模拟数据来测试算法（事实上，这比真实数据的覆盖范围更广）。生成数据的步骤请参阅第 4.2.1 节。

使用 Python 生成数据，并将其存储为纯文本文件，供 Go 语言测试程序读取。



```
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

Divide-and-Choose 算法是在两个代理之间分配不可分割物品的一种合理而直观的方法。Divider 是一个单独的代理，它根据自己的估价将物品分成两组。另一个代理人，即 Chooser，则选择自己喜欢的一组，剩下的一组留给 Divider。**这种方法确保了 envy-free，因为选择者得到了它最喜欢的一组，而进行分割的 Divider**

则认为两组的价值基本相当。Divide-and-Choose 算法在解决争端和分配资源方面非常有效，因为它能确保双方都得到公平的解决方案。

5.2.1 算法步骤

1. 首先将两名参与者随机分为 Divider 和 Chooser。
2. 确保效用最大化的唯一方法就是 Divider 尽可能均匀地分配两组。
3. 利用 bit-masking 技术高效地遍历所有情况，并将项目分为最佳 A 组和最佳 B 组。
4. Chooser 会选择对自己最喜欢的那部分，而把另一部分留给 Divider。

5.2.2 算法优化

```go
// 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

该算法使用 bit-masking 来按位运算，以组合所有项目组合的可能性，这是最快的实现方法之一，大大提高了算法的性能。

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

就像算法步骤提到的那样，算法通过 bit-masking 操作尽可能平均两组的效用。以 5 个物品为例，我们可以观察到，该算法总能找到具有最低效用的物品，并将其添加到相对劣势的一侧。

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

在 *Normal* 示例中，算法指定 Alice 为 Divider，并从 Alice 的角度尽可能均匀地将物品分成两组，使两组之间的效用差异尽可能小（在本例中，经过 bit-masking 操作后，两组之间的效用差异为 0）。然后，Chooser Bob 选择它更喜欢的部分。

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

算法似乎遇到了强大的对手。它不仅遇到了完全相同的偏好输入，而且还遇到了一个比其他项目具有压倒性优势偏好的物品（两个 Agents 相比于其他的物品，都更喜欢 Item5）。不过，DC 做得很好，它没有坚持平均分配物品的数量，而是平均分配物品的价值，这在这方面是有道理的。

测试用例参考 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

通过三种不同的 N 和三种不同的输入偏好模式，以及 500 个测试案例，对算法进行了评估，从而得出了上述图表。这些图表将作为证据在第 5.6.2 节中详细讨论.

5.3 Adjusted-Winner

Adjusted Winner 算法也是通常用于不可分割物品的公平分配算法之一。该算法旨在将有限的资源分配给参与者，同时尽可能满足公平原则。具体做法是根据参与者对物品的相对偏好决定的优胜者的价值，将物品分配给参与者。为确保公平，该算法使用了一个调整系数，以反映分配项目对其余参与者的影响，并相应调整他们的估值。在本实现方案中

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

并且，

$$adjusted\ valuation\ -=\ af\ *\ n'$$

where n' is the number of items this participant has been (pre)-allocated.

此算法保证了两个公平准则 (fairness principles)：“independence” 和 “envy-freeness”. Independence 是指算法在每次分配前都会根据该参与者已被分配的物品重新计算其调整因子以及对分配物品的估值 (adjusted valuation)。确保 envy-freeness 始终意味着其他参与者的分配效用不会高于他们现有的分配效用。换句话说，没有人会嫉妒别人。同时算法也将一定程度上考虑 PO。

5.3.1 算法步骤

1. 计算（初始）调整因子。
2. 根据经调整系数调整后的评估结果，将每个项目分配给一名参与者。
3. 检查分配是否 envy-free，如果是，则分配完成，并尽可能满意 PO。如果不是，则进入下一步。
4. 取消分配给未通过 envy-free 检查的参与者的所有项目，并重新进入分配阶段。

5.3.2 算法优化

遗憾的是，如果使用固定的调整系数，AW 并不总能找到公平的分配。本实施方案借鉴了模拟退火（simulated annealing）的思想，因此在经过一定次数的迭代后，如仍未找到合适的分配，这时调整因子就会进行小幅调整，以实现算法的完备性（所有项目都应分配给所有参与者），这也是算法上的一种折衷与妥协。

```
// 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

使用打乱机制（shuffling mechanism）还有助于避免分配过程中可能出现的偏执（biases）。

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

在 *Normal* 输入模式和 *Similar* 输入模式下，AW 都能发挥适当的作用。与 DC 相比，AW 甚至获得了更大的社会福利 (social welfare)[1]。该算法首先根据参与者的最高估价（尽可能满足 PO）分配物品，然后检查分配是否无嫉妒，当每个人都不

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

嫉妒他人的分配时结束分配。

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

与 DC 算法相比，AW 算法在 *Tie* 模式偏好方面可能会遇到一些困难。AW 算法根据参与者的最高评价来分配项目，但面对 Tie 输入模式，它只能暂时按一定顺序分配参与者。如果使用固定的调整系数，无嫉妒过程也可能陷入难以突破的窘境。为了确保算法的完整性，在达到一定的迭代阈值后，算法会进入退火阶段，在此阶段，调整系数逐渐降低，僵局开始打破。然而，在这一阶段，分配可能会变得随机，并高度依赖于参与者对项目的评价。因此，在处理对 *Tie* 模式的偏好时，AW 可能不如 DC 有效。

测试用例参考 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

通过三种不同的 N 和三种不同的输入偏好模式，以及 500 个测试案例，对算法进行了评估，从而得出了上述图表。这些图表将作为证据在第 5.6.2 节中详细讨论。

5.4 Round-Robin

Round-Robin（RR）算法是另一种广泛使用的不可分割物品分配算法。其核心思想是通过按预定顺序循环分配物品给每个参与者来实现公平分配。具体地说，该算法以循环的方式遍历参与者，并根据参与者的评价尽可能分配出最中意的物品。由于参与者在每个循环中至少得到一个物品，因此很容易平衡分配物品的数量。

RR 算法天然满足 EF1。因为该算法的工作方式是，每个参与者分配到的项目最多比优先选择较少的项目差一个。同样，它也满足了独立性，因为项目的分配只取决于参与者对项目的评价和排序机制，与其他参与者的行为无关。

5.4.1 算法流程

1. 按一定顺序排列排列参与者或将他们打乱顺序（shuffling）。
2. 按顺序迭代参与者，并尽可能分配他们最喜欢的项目。
3. 继续执行，直到所有项目分配完毕。这种分配是独立的，且满足 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

很容易看出，顺序是从 Alice 开始，然后每个人分配到自己当前喜欢的物品，直到所有物品分配完毕。

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

还是以 *Normal* 输入模式为例，算法从 Alice（事实上顺序是随机的，Bob 可能需要更多运气）开始，选择她最喜欢的未分配物品。然后轮到 Bob 选择，依此类推..... 直到所有物品都分配给所有参与者。

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

同样，RR 和 AW 一样，也没有很好地像 DC 一样很好的考虑 *Tie* 输入模式。在 *Tie* 输入模式下，第一个被分配的参与者将拥有巨大的优势，因为它总是能获得公认的最有价值的物品。如果这个最有价值的物品在评估中占据压倒性优势，例如，一栋房子与一堆杂物，那么使用 RR 的结果就是第一个被分配的参与者在数量上获得绝对领先。当然，在现实生活中是绝对不可能出现这种情况使用 RR 算法的。

测试用例参考 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

通过三种不同的 N 和三种不同的输入偏好模式，以及 500 个测试案例，对算法进行了评估，从而得到了上述图表。这些图表将作为证据在第 5.6.2 节中详细讨论。

5.5 Envy-fairness

EF1 是 EF 的折衷。也就是说，当发生妒忌时，在被妒忌的一方被拿走一件物品后，妒忌就消失了，这样的情况仍被认为满足无妒忌，即 EF1。

这种实现方法既考虑了 EF1 的思想，又更加现实。通过跟踪分配给两个参与者的物品数量，当任何一方比另一方拥有的物品多于两件时，就需要交换物品以实现公平。与 AW 算法一样，初始分配将基于相应参与者对物品的最高评价，但需要注意的是，该算法更倾向于保持分配物品数量的平衡，这一点与 AW 算法不尽相同。

5.5.1 算法步骤

1. 根据参与者的最高评价分配物品。
2. 如果偏好相同，则分配给目前收到物品较少的参与者。
3. 当任何一位参与者比另一位参与者分配的物品多于两件物品时，就会触发转移（transfer），以确保平衡。
4. 弱势参与者拿走一件相比优势参与者更喜欢的物品。如果没有这样的物品，则拿走优势参与者最不值钱的物品。

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

在 EF1 算法的这一实现中，该算法试图平衡双方被分配物品的数量，在三种偏好模式中找到最优分配。这里有两层保障措施，在分配物品时，如果有多个参与者对物品的评价相同，则会优先考虑目前收到物品最少的参与者。此外，如果某个参与者收到的物品过多，就会触发转移（transfer）以保持平衡。在触发转移之前，这种算法可以被视为一种追求最大 PO 的贪婪算法，而在触发转移之后，这种 EF1 公平性准则就得到了保证。

测试用例参考 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

通过三种不同的 N 和三种不同的输入偏好模式，以及 500 个测试案例，对算法进行了评估，从而得出了上述图表。这些图表将作为证据在第 5.6.2 节中详细讨论。

5.6 Overall Analysis

5.6.1 外部测试 (Fairallol VS. Fairpy)

在外部测试中，Fairallol 将与 Fairpy 以 Round Robin 算法进行浅层和深入测试。在浅层测试中，它用于验证两个公平分配库的行为是否一致。

```
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)

在深度测试中，Fairpy 将采用与 Fairallol 相同的方法进行测试，但只关注运行时间指标（因为这两个库在 RR 算法上的行为是一致的）。

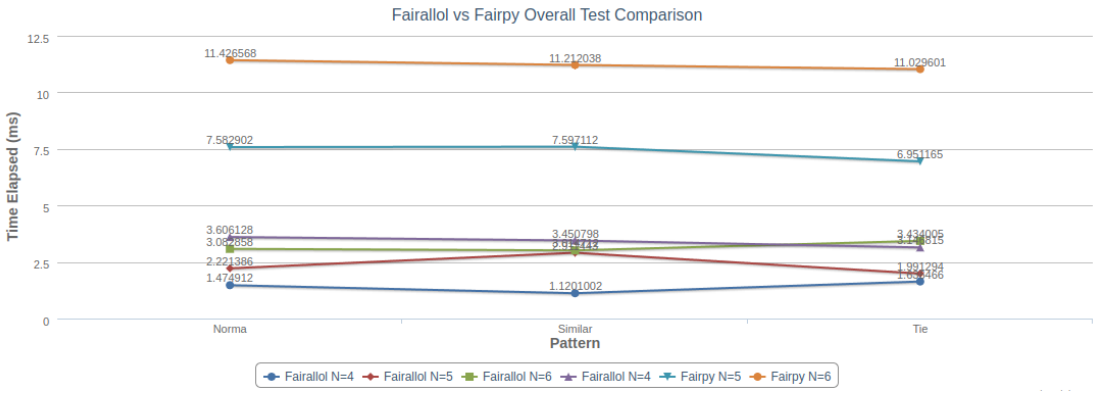


Fig. 5.9: Fairallol behaves the same as Fairpy and runs nearly three times faster.

Fairpy 的深入测试证据参考 Appendix A.2

在使用与 Fairallol 相同的测试集（每个偏好输入模式有 500 个案例）对 Fairpy 进行测试后。**发现 Fairallol 不仅具有与 Fairpy 相同的行为，而且速度提高了近 3 倍**.

5.6.2 内部测试 (Fairallol)

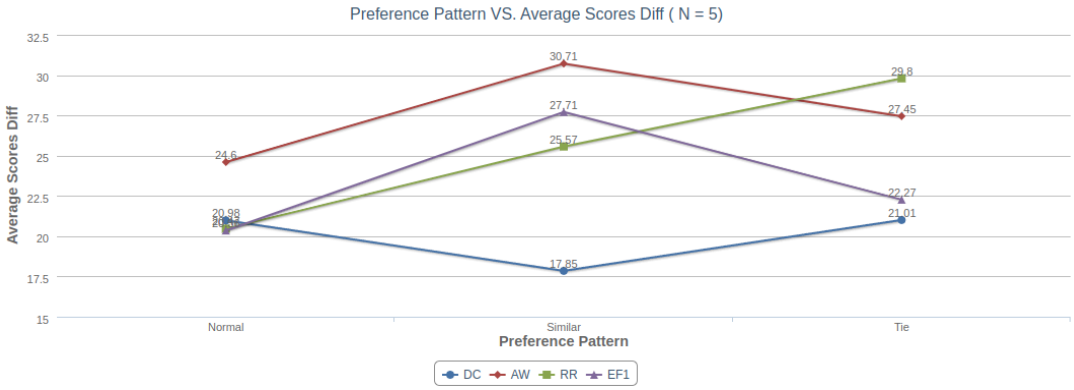


Fig. 5.10: Preference Pattern VS. Average Scores Diff (N = 5)

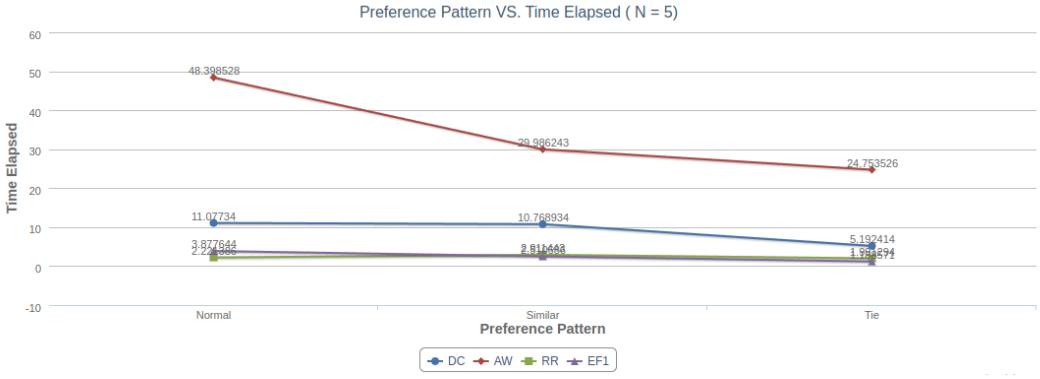


Fig. 5.11: Preference Pattern VS. Time Elapsed (N = 5)

更多证据参考 Appendix B

RR 算法可以被认为是相对简单的算法，但作为一种公平算法，它还具有额外的现实意义。简单的实现方式使其基本上具有最高的运行效率，并意味着一定的公平性标准例如 EF1 等。然而，值得注意的是，RR 的分配更倾向于自己的机制，而不是分配项目的设定或参与者对项目的评价，这导致它在处理非一般偏好模式 *Similar* 和 *Tie*）时达不到公平标准。

AW 算法的性能似乎最差，耗时最长，获得的效用差异似乎也很大。然而，耗时长的原因可能是前期使用了一个固定的调整因子，如果在前几次尝试中没有获得合理的分配，算法必须迭代到某个阈值后才会对调整因子进行模拟退火（simulated annealing）处理，而这个阈值并不容易确定，在当前的实现中，它被硬编码为在 n=1000 时模拟退火。这个值可能不适用于测试集。误差大的原因可

能是 AW 真正考虑的是其他意义上的无嫉妒，而不是最小化分配项之间的差值，它产生的所有最终分配都是无嫉妒（EF）的，而这在一些已实现的算法中是缺失的。

EF1 看起来并不很糟糕，但事实上，它在大多数时候都是一种贪婪的算法。虽然 EF1 算法能保证每个参与者都能得到一件自己喜欢的物品，但它并不能保证分配是帕累托最优的。这是因为 EF1 算法只考虑了每个参与者的个人偏好，而没有考虑整个分配方案的效率和整体福利（social walfare)。因此，即使每个参与者在 EF1 算法下都感到满意，整体分配方案对所有参与者来说也未必是最优的。

与 EF1 算法相比，DC 算法更具有全局性，其思想非常简单而实用。如果只有两个参与者参与公平分配物品，那么最简单有效的方法就是 Divide and Choose。为了使双方利益最大化或实现公平，分者应尽可能保持两组物品的平均，因为选者必然会选择更好的一组。DC 算法是在两个参与者之间分配不可分割物品的一种公平而有效的方法，它的简单性使其易于理解和在现实场景中实施。然而，当涉及的参与者超过两个，或者参与者的偏好不明确时，这种算法可能就不适用了。幸运的是，这正是本 FYP 的主题。

总之，在两个人公平分配物品的问题上，在所实现的四种算法中，DC 算法应该是最有效、最易理解、最易实现的公平分配算法。

Chapter 6

可视化

6.1 网页接口（Welcome Page）

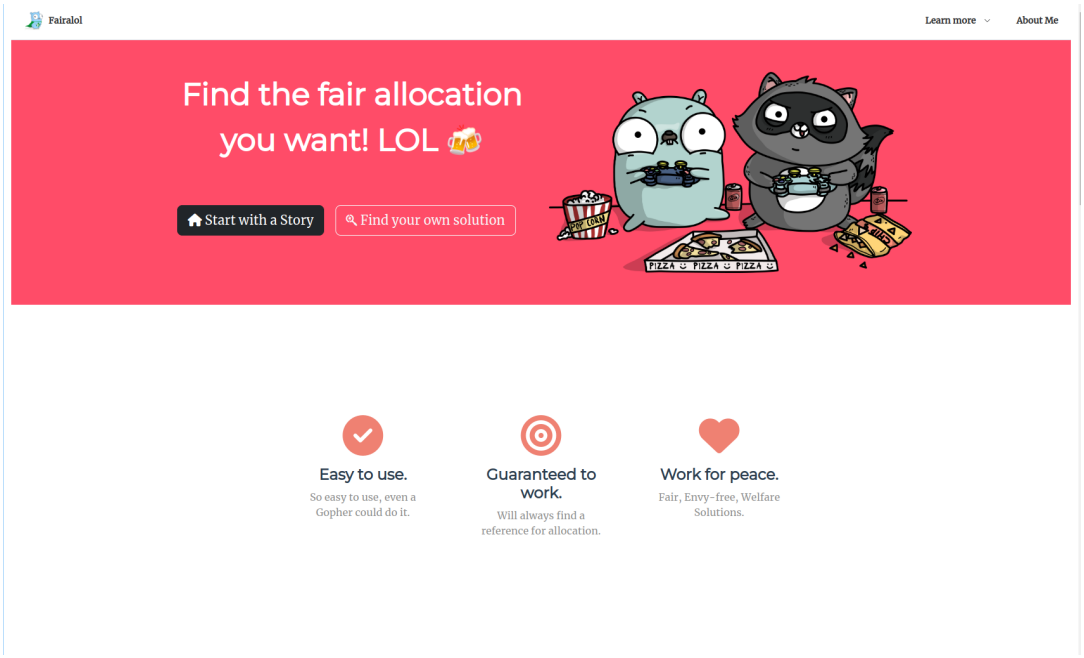


Fig. 6.1: Awesome Welcome Page

网站以红白为主色调，提供自然的互动体验，并以可爱的 Go 语言吉祥物 “gopher” 引导访问者探索和体验公平分配算法。

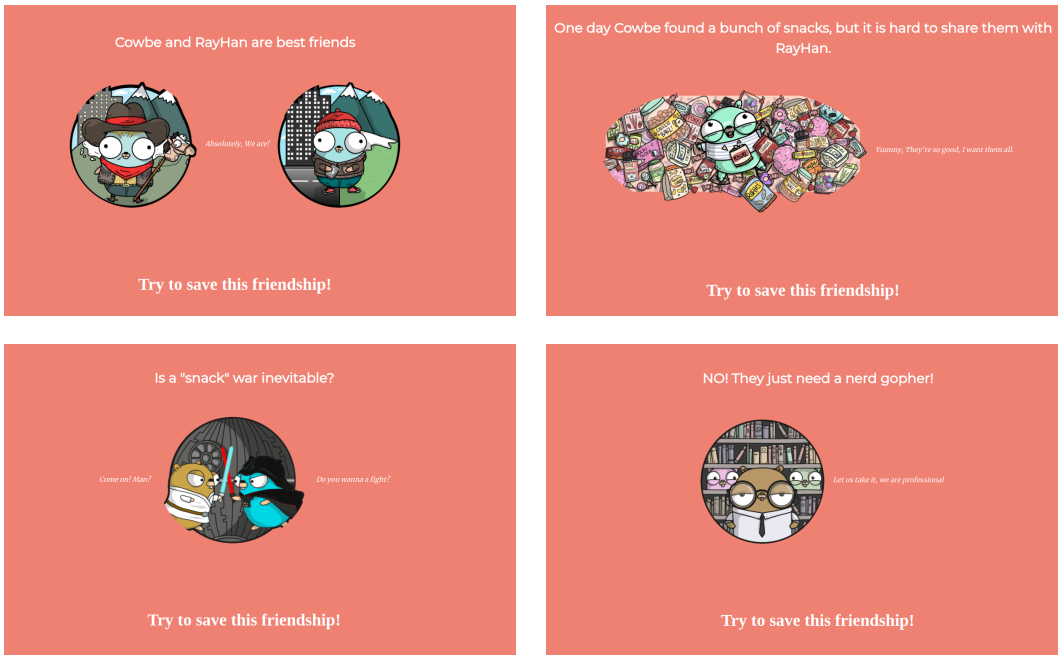


Fig. 6.2: Step into fair allocation through an interesting story

参观者可以通过一个有趣的故事体验公平分配，故事将他们带入一个需要分配算法互动的场景中 (See Fig. 6.2).

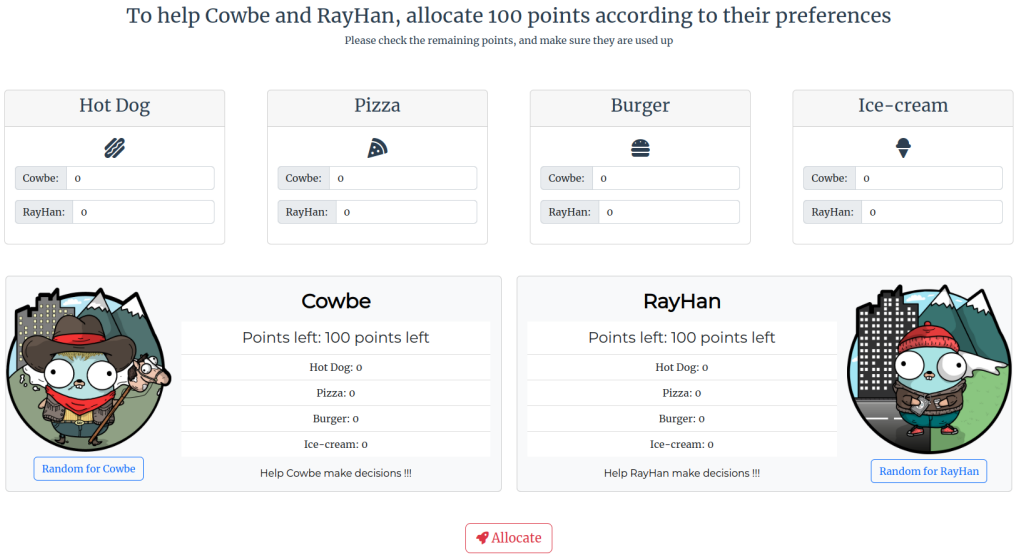


Fig. 6.3: Try the algorithm and save the friendship

之后，参观者可以在 playgroud 尝试算法，并进入公平分配的场景。最终找到自己问题的答案 (See Fig. 6.3 and Fig. 6.4).

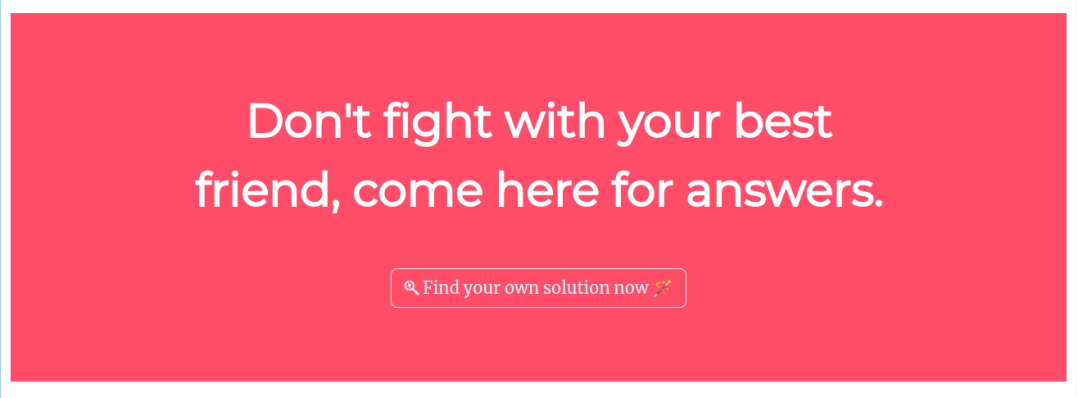


Fig. 6.4: Find your own answer and try more algorithms

6.2 网页接口 (Find Your Own Solution)

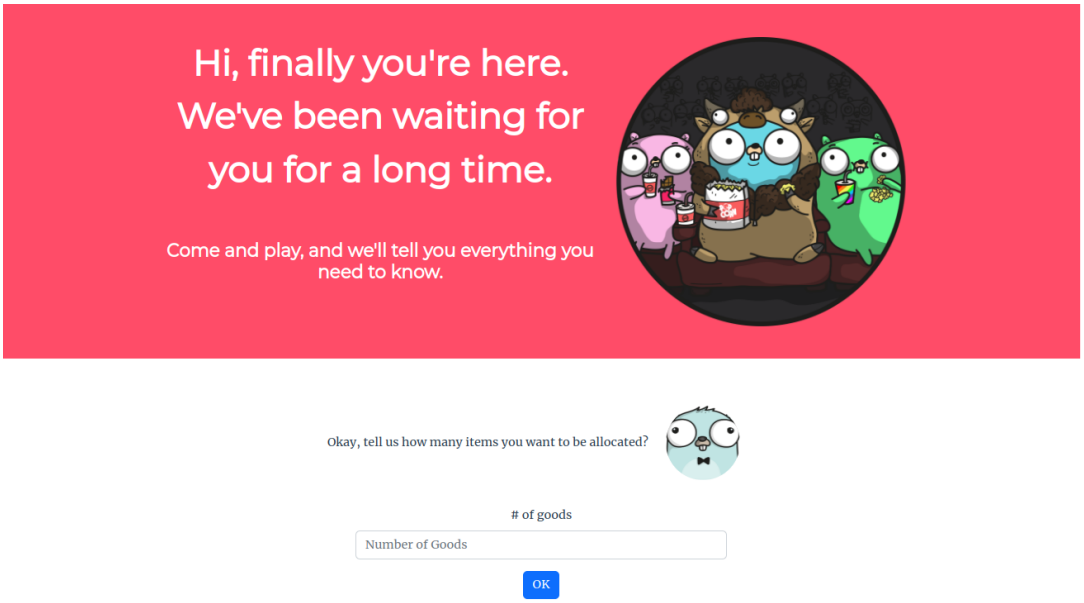
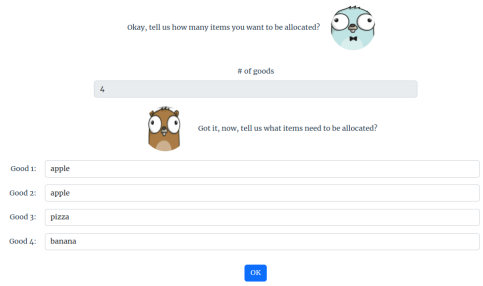
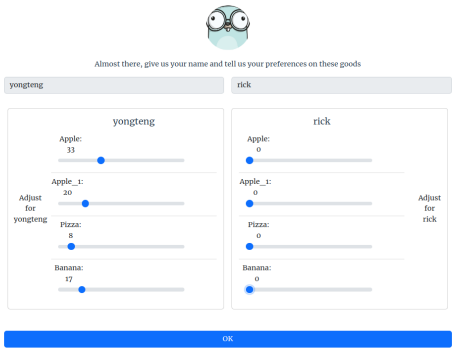


Fig. 6.5: Try more algorithms following the handy guide

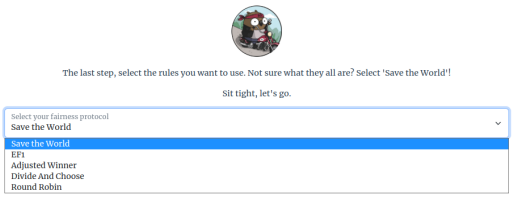
访问者可以在“Find Your Own Solution”页面尝试之前实现的算法，并将其应用到自己的实际公平物品分配问题中。访问者可以根据 Gopher 的指导一步步完善自己的设定，并逐步建立一个分配问题，最终找到自己的公平分配的参考 (See Fig. 6.6).



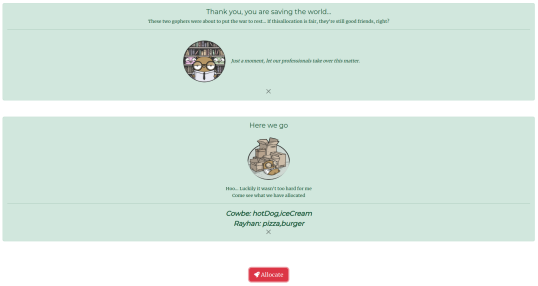
(a) Number of goods and names



(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

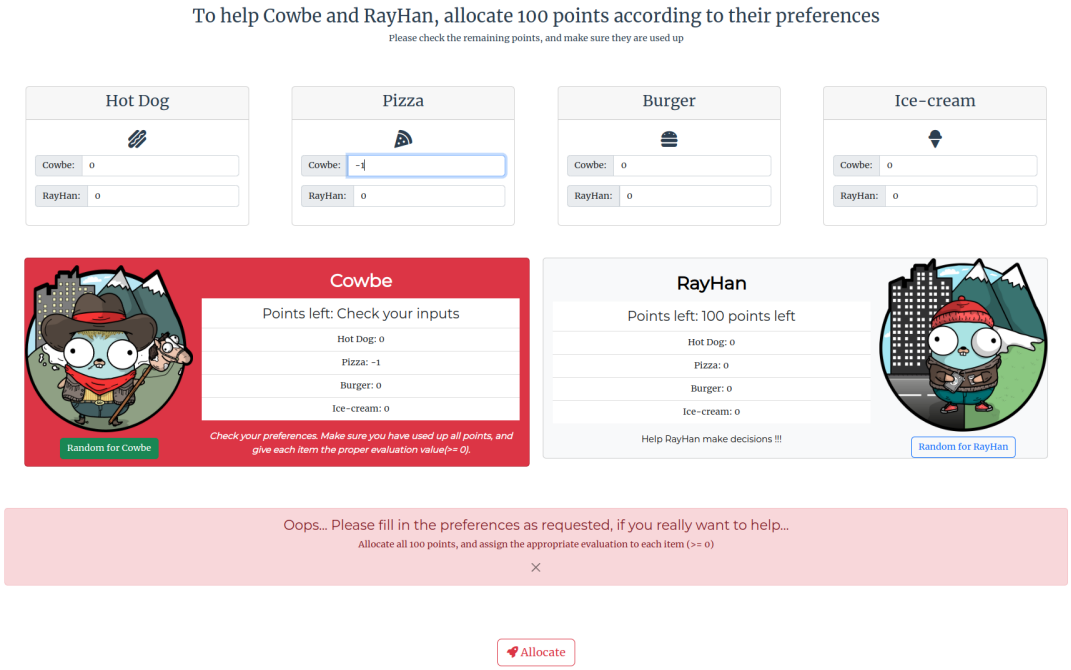
参观者确实可以利用不同的算法实现不同程度的公平自己的实际分配案例。例如，使用 Round Robin 算法，每个参与者在每一轮分配中都有相同的机会追求自己最想要的物品，而这一结果只与他们自己对物品的评价和分配机制（这里指分配顺序）有关。使用 Divide and Choose 算法，可以找到更全面的价值公平性。Adjusted Winner 算法，公平性是通过一些数学计算得出的。值得注意的是，Save the World 算法与 EF1 算法相同，都是尽可能保持分配物品数量的平衡。

6.3 健壮性以及交互友好性

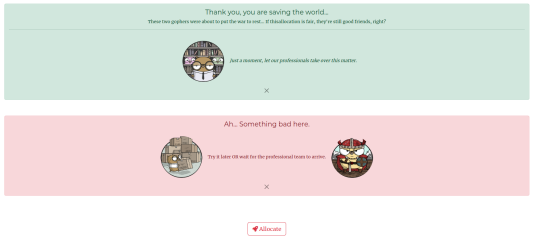
Fairallol 兼顾了美观和功能，同时还考虑到了健壮性和增强用户体验。

6.3.1 友好提示

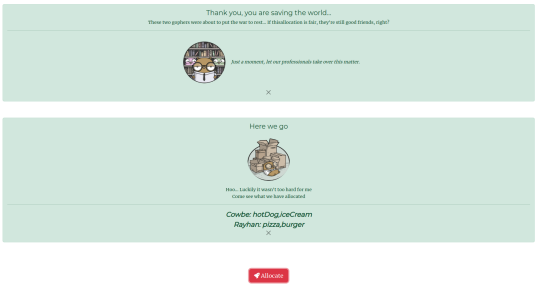
当访客输入的数值超出预期或故意干扰时，系统会显示友好的提醒，给予必要的指引或阻止访客进行下一步操作。



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



(b) Network errors or back-end crashes



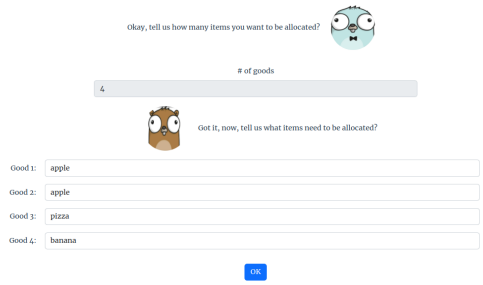
(c) Everything is going well

Fig. 6.7: Hints at the Playground

更多在 Find Solution 页面的提示示例，参考 Appendix C

6.3.2 健壮性

前面提到的四种算法将作为后端服务实现供游客探索和体验。不过，所有待分配的物品名称和参与者名称都必须是唯一的，前端应用程序要确保这一点，以保证其稳健性。



(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

更多网站健壮性示例，参考 Appendix D

6.3.3 用户体验增强

记录用户对每个待分配项目的偏好非常重要，也是整个互动过程中最重要的部分。playground 中提供了随机选项，让游客更容易体验算法。而在 Find Your Own Solution 部分，提供了一个调整选项，可根据用户的现有输入来缩放点数，以满足算法要求。

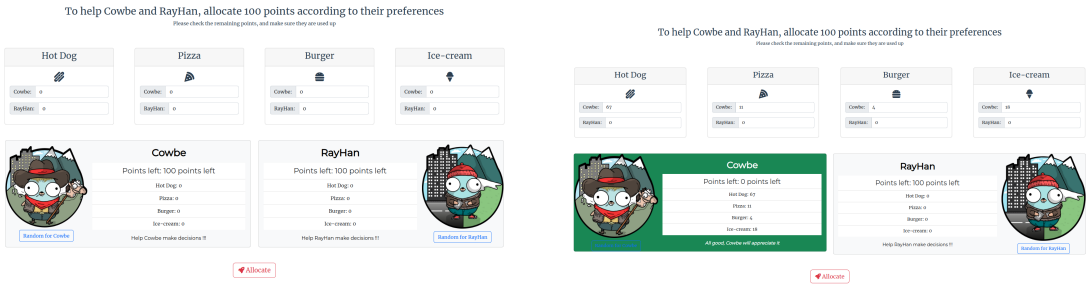


Fig. 6.9: Random option for visitors to experience the algorithm

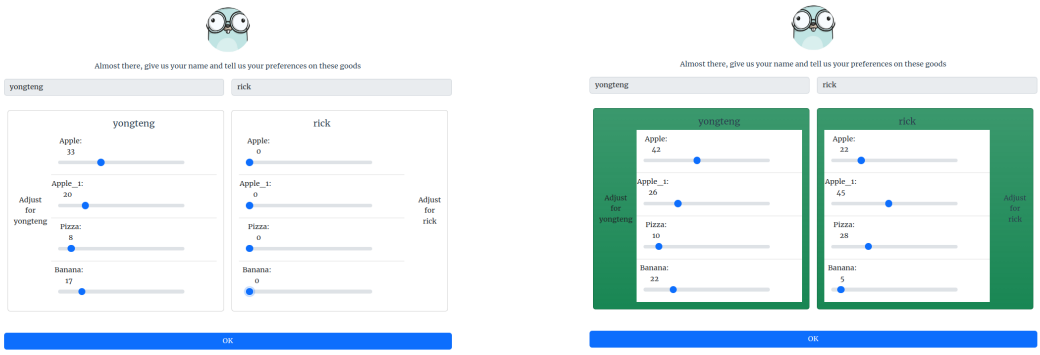
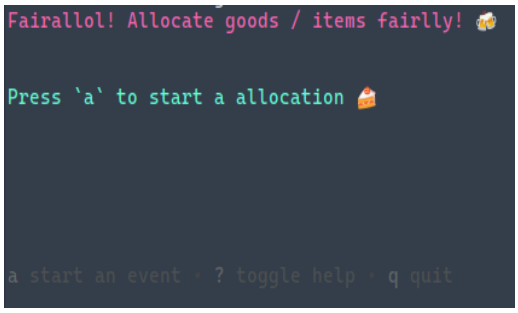


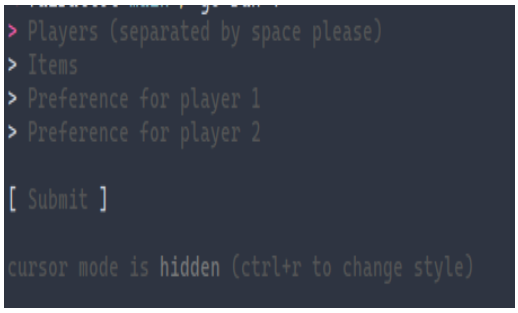
Fig. 6.10: Adjustment option makes input to satisfy requirements on existing scale

6.4 终端程序

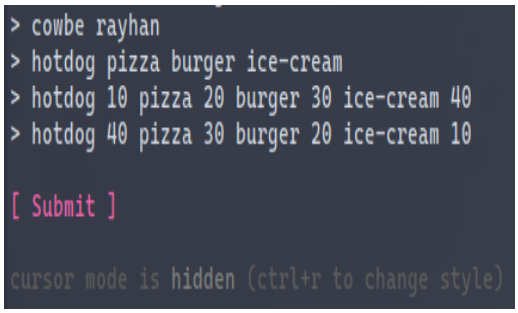
该终端小程序适合那些追求高效率、喜欢 *GRAB-AND-GO* 的用户。



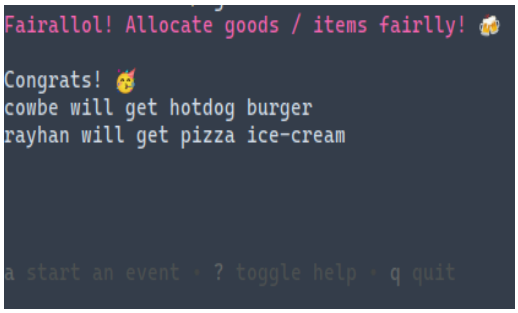
(a) Start



(b) Submit



(c) Allocate



(d) Finish

Fig. 6.11: Get an allocation in under a minute

Chapter 7

局限性和未来工作

7.1 局限性

7.1.1 算法

公平分配面临的一个主要挑战是如何解决个人的不同偏好和评价，因为一个人认为公平的分配在另一个人看来可能并不公平。就公平的含义达成共识也很困难，因为无嫉妒 EF、PROP 和效率等概念可能有不同的优先级。此外，找到理想解决方案的复杂性会随着参与者和所涉及项目的数量而增加，因此在大规模场景中实现公平具有计算上的挑战性。

尽管本 FYP 的主题只涉及探索两个参与者之间物品公平分配的问题以及广泛应用的方法，但仍存在一些局限性。

- **算法实现妥协**

  实现算法有多种方法。RR 和 DC 算法由于简单实用，可以非常容易高效地实现。

  然而，像 AW 和 EF1 这样的算法因为没有统一的标准而难以实现。在 FYP 的实现中，AW 算法不能保证不可剥夺性，这意味着分配给参与者的项目可能会被剥夺并参与下一轮分配，这违反了经典 AW 算法算法所保证的性质。

  至于 EF1 算法，由于该算法试图通过在一开始就为参与者分配最高的项目偏好来保证 PO，因此该算法可被视为一种贪婪算法，它试图在局部实现最优解，但往往也因为是局部最优解而达不到目标，即公平以及 PO。虽然这种算法保持了一定的公平性，分配的项目数量均衡，但在某些情况下也存在弊端，请参考第 5.5.3 节。

- **算法完备数据与现实数据之间的差异**

虽然没有现实数据，但可以生成测试集，并以组合方式测试算法（甚至获得更大的覆盖率）。然而，由于数量庞大，而且没有现实的预期结果可供比较，算法的效率和获得的效用只能通过第 4.2.1节中提到的标准来评估。但这样的测试缺乏现实生活中分配的意义。

7.1.2 用户接口

尽管目前的网站具有良好的感觉和外观，具有一定的稳健性和良好的互动性，但仍然存在一些局限性。

- 网站对方框模型的控制有些欠缺，在动态更改提示时，有时文字会使 box 模型变大，造成视觉上的不一致。
- 虽然界面是响应式的，这意味着网站可以在电脑和手机上正常显示，但由于缺乏更细粒度的组件控制，界面有时会显得有些 “拥挤”。

7.1.3 项目管理

事实上，作为在这一领域做一些实验的探索者是相当令人激动的。公平分配问题是一个小众但与每个人都息息相关的问题，自第二次世界大战以来就一直有人在研究，最近还成为诺贝尔奖得主的研究课题。主流算法开发通常使用 Python 编程语言，如 Fairpy。许多注重效率的算法使用 C/C++ 进行更底层的控制。然而，Go 作为最有前途的新编程语言，在这方面的探索却较少。Fairalol 作为此类尝试的探索者，得益于 Go 语言的固有优势，在实现相同行为时比 Python 快约三倍。然而，Fairalol 在工程设计上仍有缺陷。例如

- 只管理简单的项目结构。当初始项目规模不够大时，最好保持简单，但这可能会为日后代码膨胀带来隐患。
- 缺乏逻辑抽象。理想的做法是定义一个接口，实现该接口，并让结构拥有自己的方法，从而确保结构受到某些约束，以确保逻辑得以保持。目前的做法是让结构直接调用方法，这为预先测试和修改提供了方便，但绝不是一种好的规范。
- 代码冗余。当前的测试代码中有大量未正确提取的冗余代码。
- 入门文档。应提供便于初学者使用的文档和贡献指南，以便交流和激励贡献者。

7.2 未来工作

这次 FYP 之旅不仅在公平分配这一经济学、博弈论和社会学的关键问题上，而且在软件开发领域，都是一次激动人心的探索之旅。

在算法方面，公平分配的基本原则和标准已经得到了一定程度的论证。之后，可以利用线性受限的计算方法对分配进行更全局性的探索，而不是通过贪婪的数学方法和机制来实现目标。

在整个软件开发过程，前端已经设计完成并实现，后端网络服务也从无到有，前后端集成也顺利完成。但是，在完善网站界面方面仍有改进的余地。此外，在征得访问者同意或对数据进行适当脱敏的情况下，可以记录相关的真实世界数据，访问者可以就分配过程留下反馈意见。这些反馈可以纳入算法，以优化迭代分配过程。

本次的 FYP 项目不仅是作者的一次尝试和创新，也是一个开创性的、完整的探索过程，可供他人参考和改进。在作者对代码进行模块化重构并撰写详细文档后，Fairallol 网站的源代码保持开源，读者可以调取代码，构建并访问自己的公平算法进行实验。

Fairallol 后端代码作为 Go 中公平算法的早期探索者，试图为志同道合的同行提供学习和改进的机会，作者将积极开发和改进代码，并鼓励潜在贡献者为开源社区和正在进行的公平算法讨论做出贡献。

Chapter 8

总结

在这篇 FYP 中，作者探讨了有两个参与者参与的不可分割物品的公平分配问题。作者探讨了在这种情况下最广泛使用且最有效的四种方法，即 Divide And Choose（DC）、Adjusted Winner （AW）、Round Robin （RR）和 Envy-freeness up to one（EF1）。经历了前端网页设计和开发、后端部署以及前后端交互的完整过程。所探索的四种公平分配算法被移植到了一个现代的、易于使用的、强大的启发式网页界面上，此外还可以访问另一个终端界面。但这里仍然存在一些局限性，因为可以在适当的条件下收集界面访问者及其评论的真实数据，以反馈给算法，从而反复优化算法，实现更切合实际的分配。此外，通过可以尝试使用线性限制（Linear constraint）等计算方法，可以在后期实验中对公平分配进行更全局的考虑，以实现现实的公平性（尽管该问题可能是 NP-hard，但在某些设置下，可以找到多项式级的近似值），而不是使用数学方法和机制来实现目标。

感谢本次的 FYP 为作者提供了这个探索公平分配算法世界、并锻炼论文阅读与算法实现的宝贵机会。公平分配是一个小众研究领域，且有一定门槛，几乎不可能在互联网上找到清晰的，新手友好的入门档案。然而，这种情况促使作者阅读文献，研究数学表达式和算法的内在含义，培养阅读伪代码并尝试实现它们的能力。这为今后可能的研究道路打下了坚实的基础。此外，此次 FYP 还真正实现了我四年来的梦想。真正建立一个用于研究的前端和后端，以及一个向公众开放的完整界面，这也与我的职业规划息息相关。感谢导师一直以来的指导和鼓励，作者才得以做到这一点。

参考文献

[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.

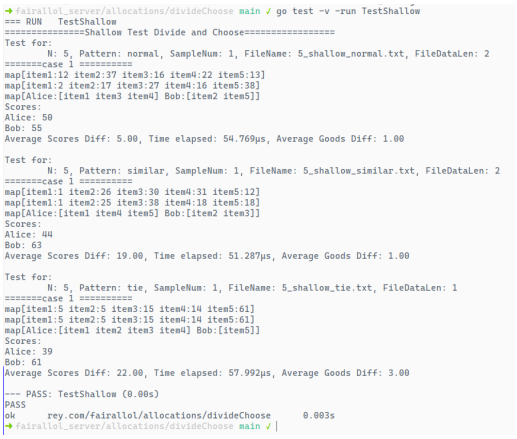
[22] H. Varian, "Equity, envy and efficiency," *J. Econ. Theor.*, vol. 9, p. 63–91, 1974.

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

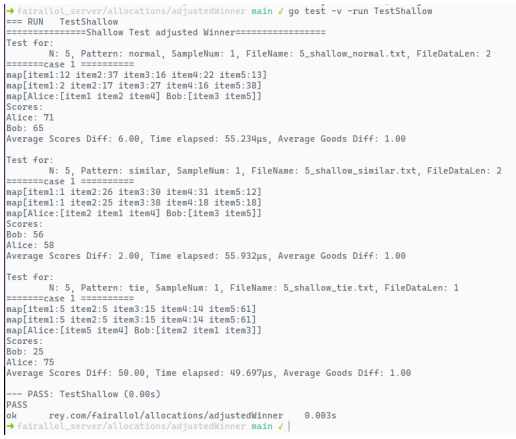
[24] A. Procaccia and J. Wang, "A lower bound for equitable cake cutting." in *EC 2017 - Proceedings of the 2017 ACM Conference on Economics and Computation*, no. EC 2017 - Proceedings of the 2017 ACM Conference on Economics and Computation, pp. 479–496 – 496.

Appendix A

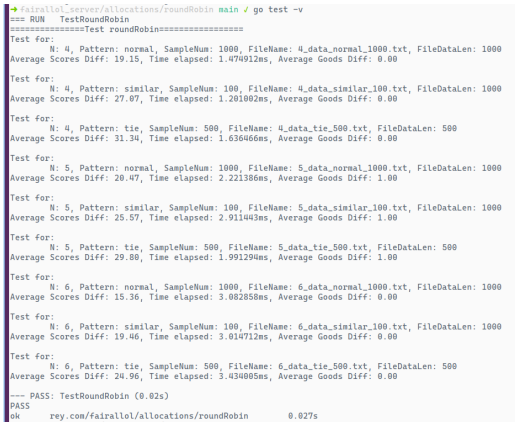
浅测试



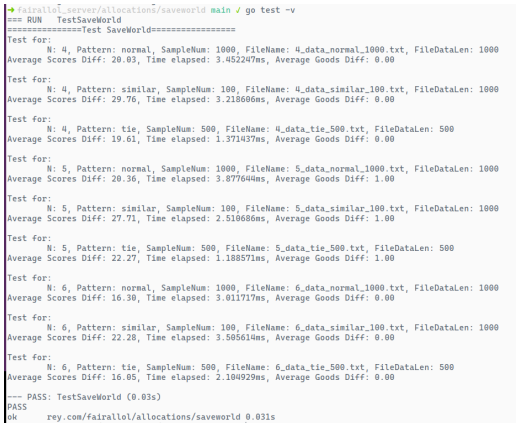
(a) Shallow Test for DC



(b) Shallow Test for AW



(c) Shallow Test for RR



(d) Shallow Test for SW

Fig. A.1: Shallow Test

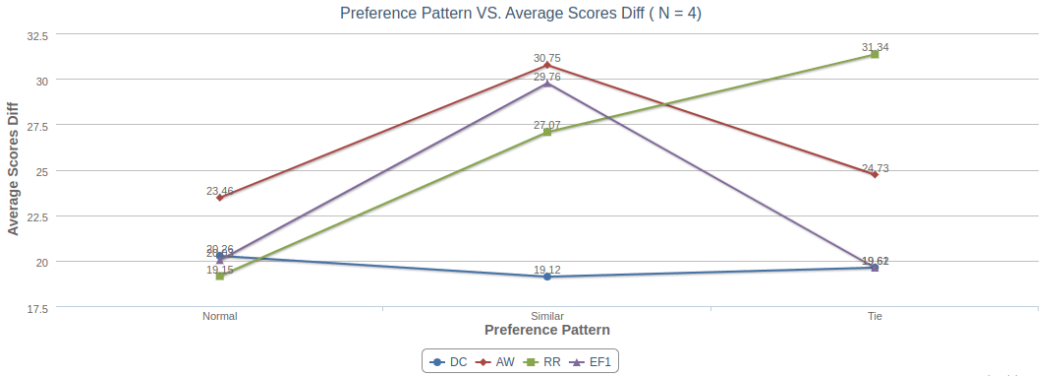
```
Test for:
        N: 4, Pattern: normal, Time elapsed: 0.03606128692626953
Test for:
        N: 5, Pattern: normal, Time elapsed: 0.07582902908325195
Test for:
        N: 6, Pattern: normal, Time elapsed: 0.11426568031311035
Test for:
        N: 4, Pattern: similar, Time elapsed: 0.034507989883422285
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]
```

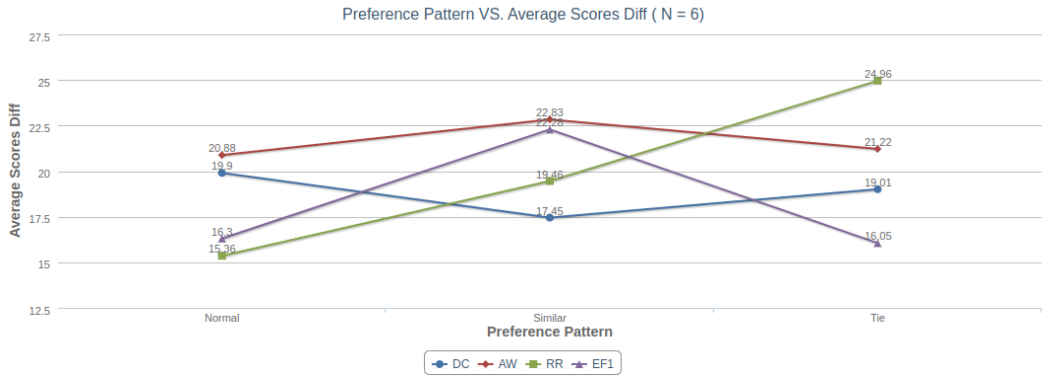
Fig. A.2: Shallow test for Fairpy (The Same behavior as Fairallol)

Appendix B

总体分析

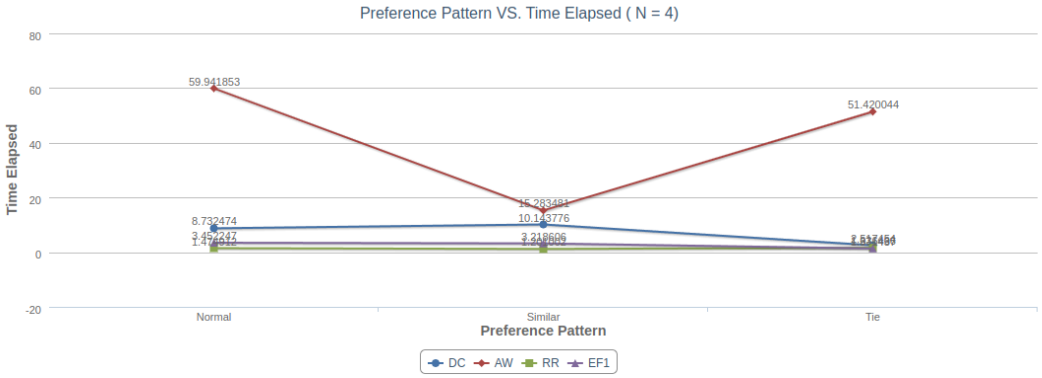


(a) Preference Pattern VS. Average Scores Diff (N = 4)

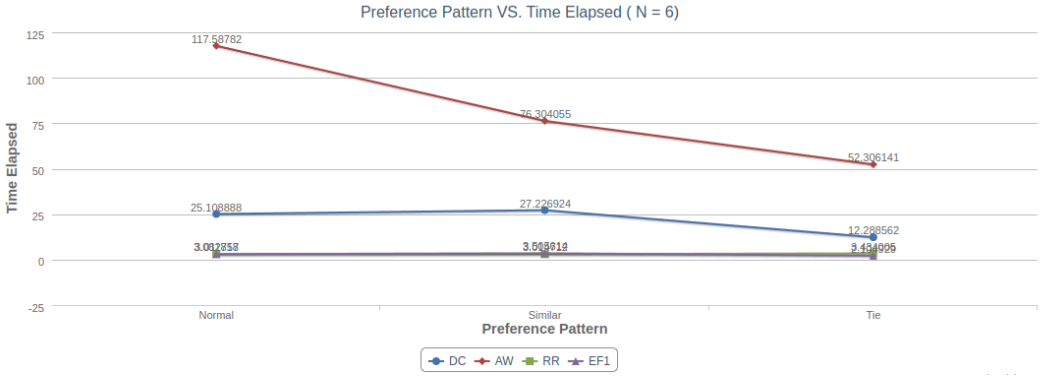


(b) Preference Pattern VS. Average Scores Diff (N = 6)

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



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

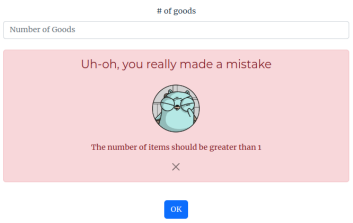


(b) Preference Pattern VS. Time Elapsed (N = 6)

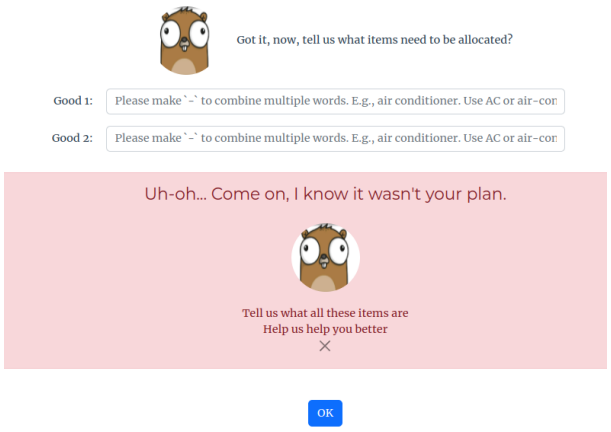
Fig. B.2: Overall Analysis - Pattern VS. Time Elapsed

Appendix C

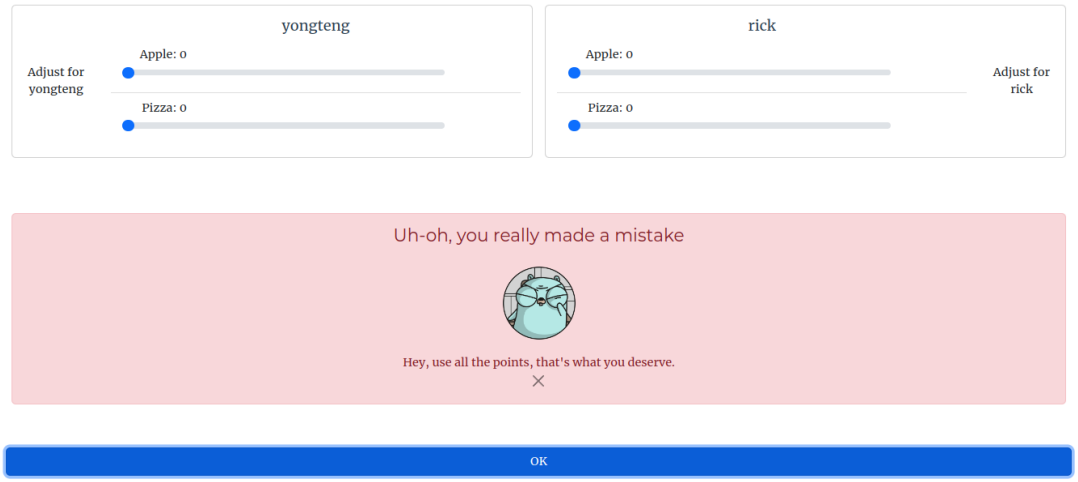
更多 Find Solution 页面中的提示



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

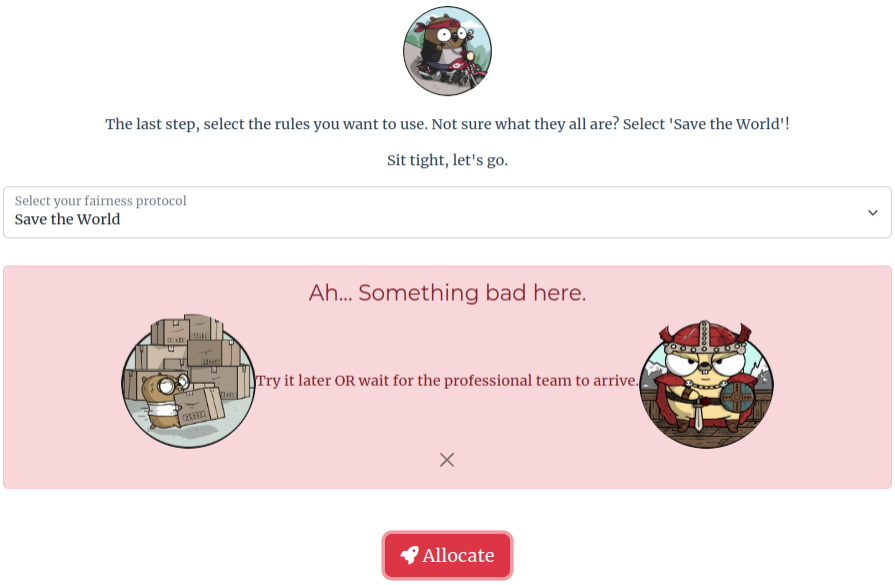


(b) Network errors or back-end crashes

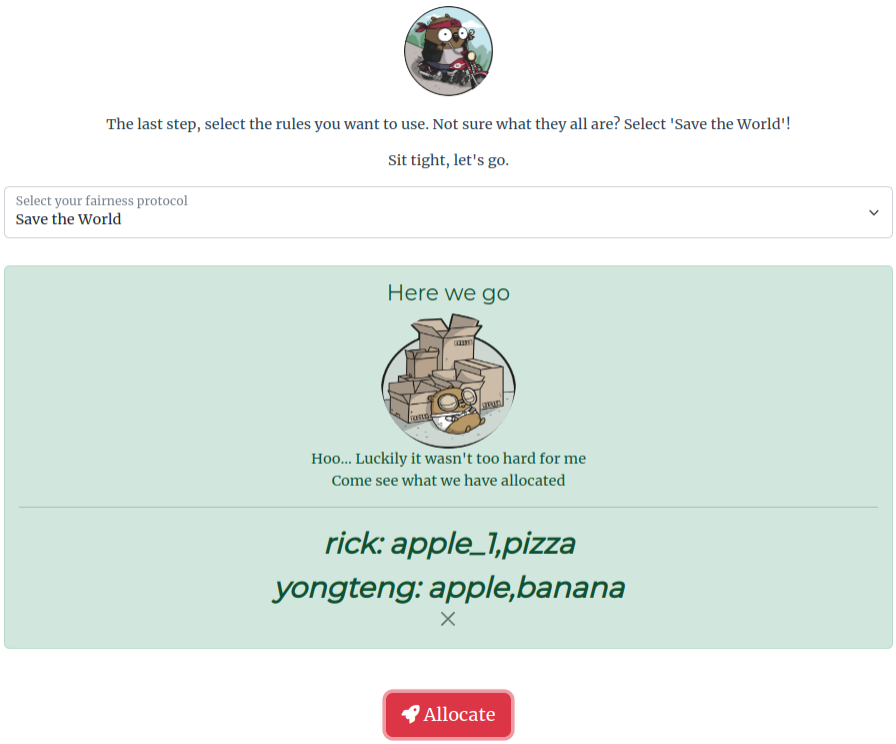


(c) Everything is going well

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

网站健壮性例子

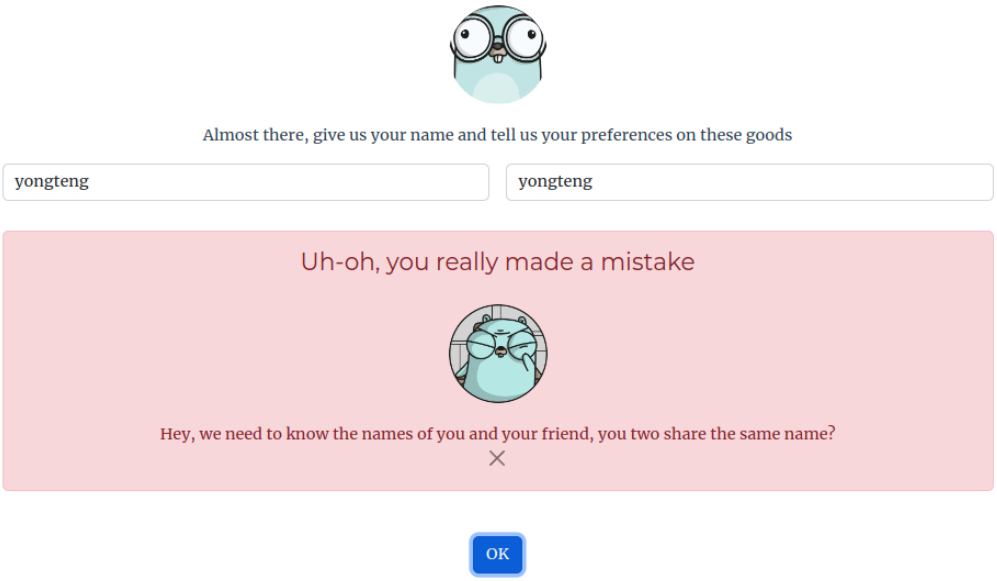


Fig. D.1: Front-end application intercepts duplicate names

Appendix E

最后想说的话

公平分配是一个从二战开始就在研究的话题，它在经济，数学以及计算机科学领域均有大量的学者在不同的方面进行研究，至今，这个问题还不能很好的解决现实中的问题（甚至以后很长的一段时间也可能无法解决）。可能受到算力与现实的复杂场景的影响，算法提供的分配方案可能无法满足所有人对公平的定义，例如，它不能预防人们在一开始就隐瞒对自己的物品的喜好，无法防止人们在深知算法机制后，对不熟悉“规则”的对方的“欺骗”（cheating）。参考纽约大学维护的“Adjusted Winner”算法的网站，一个国家可以使用算法合理的在巴拿马运河条约谈判中获得更多有利的资源（两个国家间资源的公平分配），一个对丈夫不了解的妻子也可能被完全了解自己的丈夫通过隐瞒他自己对物品的偏好而获得更多的财产（两个小人物之间的公平分配）。

“程序正义 Procedural justice”与“最终正义 End justice”一直是具有争议的话题。在上面的例子中，有些人们会站在妻子的立场上而谴责丈夫——站在“最终正义”的立场上，丈夫不应该通过欺骗算法而谋取更多的利益。而一些“不近人情”的人们可能维护丈夫，“他只是合理利用规则”。这样的两种正义就决定着计算机可能不能给出一个“完美”的答案，人们对待公平的看法可能不同。

如果你打开网站就能从一个故事里进入这个公平分配的话题，两个小地鼠（Gopher，Golang 的吉祥物）正要因为分配四个零食而发生争吵，好吧，他们想要通过一个地鼠专家（一个 Golang 程序）来帮他们解决这个问题，如果一切顺利的话，他们应该每人都会获得两个零食（可能是一个填饱肚子的大家伙和一个甜点）。但一个傻地鼠完全可能因为自己的失误而导致自己只获得一个甜点而其他三个零食都被它曾经的小伙伴带走，这段友谊因此而破裂。如果使用“Save the World”算法，就会避免这样的事情发生，这是我在算法实现过程中的选择，两个好朋友不应该因为零食的问题而冷战，哪怕一天。

选择“公平分配”问题作为 FYP 绝对是一个奇妙的体验，尽自己所能去探索这个领域，但还是保持谦逊，因为我们巨人的肩膀可能太宽太广（如果你能明白我的意思）。2012 年，美国经济学家、加州大学洛杉矶分校教授劳埃德 · S · 沙普利和美国经济学家、哈佛大学教授阿尔文 · E · 罗思因为对“稳定匹配理论”的研究，这是个非常细分的领域，终于才斩获诺贝尔经济学奖。但在这个旅程中，我们可以扮演一次“上帝”，根据我们自己的对公平的理解，来决定这些 Gopher 的命运，站在“程序正义”的你，可能真的会让一个小 Gopher 伤心。期待之后的同学能在这个领域的探索中，开发出大多数都认可的公平分配算法，说不定下一个诺贝尔奖就是你!

Fig. E.1: 两只 Gopher: Cowbe (Cow boy) 和 Rayhan (Red hat)