

UNDERGRADUATE THESIS
IN MATHEMATICS AND COMPUTER SCIENCE

MAXIMIZING NASH SOCIAL WELFARE IN ONLINE SETTINGS

UNIVERSITAT POLITÈCNICA DE CATALUNYA¹



UNIVERSITY OF MARYLAND²



Author: Edgar Moreno Martínez¹

Advisor: MohammadTaghi Hajiaghayi²

Tutor: Carme Àlvarez Faura¹

Academic Year: 2022/2023

Deposit: May 2023

I would like to specially thank Suho, Max and Kiarash; with whom I have been collaborating to find the results explained in this work. I would also like to thank Prof. MohammadTaghi Hajiaghayi for welcoming me to his team. And to Prof. Carme Àlvarez for helping me find this awesome group.

I would like to thank the *Interdisciplinary Center of Superior Formation* (CFIS) for the opportunity of developing my thesis abroad and to the *Cellex Foundation* and the *Polytechnic University of Catalonia* (UPC) for the financial support, and specially to all the people in those organizations for the personal support all this years.

En la banda més personal m'agradaria agrair l'Albert i el Roger (i tota la gent que he conegut!) per fer aquests mesos a DC més divertits. I a la Sílvia per fer-ho tot una mica millor en les distàncies curtes i en les llargues.

Aprofitant, també m'agradaria agrair als matesinfos, als matesfísica, als mates-bar i als professors de la FME i el CFIS per aquests últims 5 anys magnífics. Als meus amics de Tarragona, Girona i Barcelona per guardar un raconet on tornar. A la meva família per tenir sempre preparada una casa on anar i per empenyer-me desde petit en aquest camí que és la vida.

I al Marc, per ensenyar-me que el camí, faci pujada o baixada, sigui a una classe o de festa, és una cosa divertidíssima i que s'ha de gaudir amb un somriure sempre. ▲

Abstract

In this work we consider the problem of allocating a set of T indivisible goods to a set of n agents in an online manner, with the goal of maximizing the Nash social welfare (NSW) to balance both fairness and efficiency. In the online setting, goods arrive in a sequential fashion and the value of each good is revealed only at the time of arrival, with these values generally selected by an adversary.

We examine the NSW problem within the *identical* and *binary* valuation settings, as well as mild generalizations to each. We present lower and upper bounds on the competitive ratio for each scenario. We highlight the exponential lower bound for the *binary* and general case, and good guarantees for the greedy algorithm in several settings.

In the equal setting, we find an almost tight lower and upper ratio in line with the work of Barman et al. (EC'18). We also study the setting where a large number of binary items arrive getting asymptotically tight guarantees. Lastly we use some ideas from the “bipartite maximum matching” for deriving lower bounds in the *bivalued setting*.

MSC Code 91B32: Resource and cost allocation (including fair division, apportionment, etc.)

Keywords: Online Algorithms, Allocation Problems, Nash Social Welfare, Randomized Algorithms, Competitive Ratio, Adversarial Input

En aquest treball considerem el problema *online* d'assignar un conjunt de T béns indivisibles a un conjunt de n agents, con l'objectiu de maximitzar el Nash Social Welfare (NSW) per equilibrar tant la equitat como la eficiència. En el model de problema *online*, els béns arriben de manera seqüencial i el valor de cada bé només es revela quan arriba, amb aquests valors normalment seleccionats por un adversari.

Examinem el problema quan els agents valoran els béns de manera *igual* i *binaria*, així como generalitzacions de cada cas. Presentem fites inferiors y superiors per al *competitive ratio* a cada escenari. Destaquem la cota inferior exponencial para el cas *binari* i general, i bones garanties para el algorisme *greedy* en diversos escenarios.

A l'escenari amb valoracions *iguals*, trobem un *competitive ratio* congruent amb el treball de Barman et al. (EC'18). També estudiem l'escenario on arriben un gran número de béns amb valoració *binaria* i obtenim garanties assímptóticamente ajustades. Por último, utilitzem algunes ideas del problema “bipartite maximum matching” per derivar fites inferiors en el escenari *bivaluat*.

Paraules clau: Algorismes Online, Problemes d'Assignació, Nash Social Welfare, Algorismes probabilistic, Competitive Ratio

En este trabajo consideramos el problema *online* de asignar un conjunto de T bienes indivisibles a un conjunto de n agentes, con el objetivo de maximizar el Nash Social Welfare (NSW) para equilibrar tanto la equidad como la eficiencia. En el modelo de problema *online*, los bienes llegan de forma secuencial y el valor de cada bien solo se revela en el momento de su llegada, con estos valores seleccionados normalmente por un adversario.

Examinamos el problema cuando los agentes valoran los items de manera *igual* y *binaria*, así como generalizaciones a cada caso. Presentamos cotas inferiores y superiores para el *competitive ratio* para cada escenario. Destacamos la cota inferior exponencial para el caso *binario* y general, y buenas garantías para el algoritmo *greedy* en varios escenarios.

En el escenario con valoraciones *iguales*, encontramos un *competitive ratio* en línea con el trabajo de Barman et al. (EC'18). También estudiamos el escenario en el que llega un gran número de bienes con valoración *binaria* y obtenemos garantías asintóticamente ajustadas. Por último, utilizamos algunas ideas del problema “bipartite maximum matching” para derivar límites inferiores en el escenario *bivaluado*.

Palabras clave: Algoritmo Online, Problemas de Asignación, Nash Social Welfare, Algoritmos aleatorizados, Competitive Ratio

Contents

1	Introduction	6
1.1	Allocation problems	6
1.1.1	Why NSW?	7
1.2	Online problems	9
1.3	Randomized algorithms	9
1.3.1	Ski-rental problem	10
1.3.2	Yao's Principle	11
1.4	A proper definition of our problem	13
1.5	Related problems	14
1.5.1	NSW offline maximization	14
1.5.2	Divisible items online problem	14
1.5.3	Online Santa Claus problem	15
1.6	Overview of the results	16
2	General lower bounds	17
3	Equal valuations setting	21
3.1	Greedy algorithm	21
3.2	Lower bound on deterministic algorithm	24
3.2.1	The function $\exp(\ln x/x)$	26
3.3	Small items case	30
3.4	The intrinsic difficulty	32
3.4.1	Optimal distribution	33
3.5	Randomized algorithms	36
3.5.1	Lower bound	36
3.5.2	A proposal of a randomized algorithm	37
4	Affine utility setting	42
5	Bivalue setting	45
5.1	Hardness in the offline setting	45
5.2	Binary setting	45
5.2.1	Large number of items	46
5.3	m -Bivalue setting	48
5.3.1	Online bipartite maximum matching	50
5.3.2	Relation with bipartite maximum matching	53

<i>CONTENTS</i>	5
5.3.3 Negative results inspired of random tie-breakers algorithms	54
6 Conclusion	57
Bibliography	58
A Code 1	61
B Code 2	65
C Code 3	67

1. Introduction

It is self-evident that resources in our world are finite and so is wealth in our society. Thus, how should it be distributed is a difficult question. The study of this problem comprises the whole area of economics's studies with deep implications in sociology, history and philosophy.

In recent times this kind of questions have had a translation into more abstract terms entering the fields of mathematics and computer science. This developed the whole area of (Algorithmic) Game theory. It tries to model the decisions of humans beings supposing that they were rational agents trying to maximize an utility function in a set environment. This maximization can come from competition or cooperation between them. Although its results do not always match reality (Camerer [2014]) it gives good intuitions about which incentivise can people have in certain situations.

The area developed to study two other interesting questions: how to design mechanisms that incentives agents to compete in a positive way (for the designer of the mechanism) and how to distribute a given number of scarce resources. This last question is known as allocation problems and creates a family of other questions in the way. This work focuses on an specific sub-problem of allocation theory but an introduction of allocation problems comprises the next section.

1.1 Allocation problems

Suppose that we have a set of n agents $A = [n]$ and a set of goods (or items) G , that they can not possess at the same time. Furthermore suppose that we have a function $u : A \times P \rightarrow \mathbb{R}^{\geq 0}$, where $P = \mathcal{P}(G)$ (the subsets of G), we call this function utility function and is usually noted as $u_i(p) := u(i, p)$. This utility functions reveals the value that each agent gives to obtaining an specific set of goods. There are several properties that a utility function is usually required to satisfy:

- Non-negativity: every set of items provides a non-negative utility to the agents. Although this has been the case for the vast majority of the research on the field, there is some recent works (Aziz *et al.* [2018]) that study the setting with *goods and chores* where the latter are something to avoid with negative utility.
- Positive marginal impact: adding items has a non-negative impact on the valuations, this is a very natural axiom. Formally, given a set of items S and an extra item I , for any agent i we have $u_i(S \cup \{I\}) \geq u_i(S)$.
- Marginal-decreasing utility: this asks that when adding an item to a set of items the added utility decreases when the original set has more items. Formally, if we have sets of items A, B with $A \subseteq B$ and an item I , for any agent i we have $u_i(A \cup \{I\}) - u_i(A) \geq u_i(B \cup \{I\}) - u_i(B)$.

We will also suppose that u is *additive*, this is that for $P, P' \subseteq G$ s.t. $P \cap P' = \emptyset$ we have $u_i(P \cup P') = u_i(P) + u_i(P')$ for all $i \in [n]$. This is the most simple setting but is usually realistic enough. In this work we will use *indivisible* items, this is, that we can not fraction them and allocate each fraction to a different agent. It is a wide studied topic, but there is also vast literature in the *divisible* setting from which we will borrow some results.

Now a question arises: if we are a “dictator” that decides how to allocate the items, which allocation should we prefer. In order to answer this question we can introduce a “social utility” function for a partition. Given $\{X_i\}_{i \in [n]}$ s.t. $\bigcup X_i = G$ and $X_i \cap X_j = \emptyset$ let $F(\{X_i\}_{i \in [n]}) \in \mathbb{R}$ its value. Then we want to maximize this function F . Then the big question here is how to choose F . In this work we choose what is known as Nash Social Welfare (NSW). There are also non-numeric properties that might be desirable to achieve. We discuss social utility functions and this properties in the next section.

1.1.1 Why NSW?

The question about which properties are desirable falls on the social sciences spectrum, but intuitively we can understand some of the most studied (Plaut and Roughgarden [2017]):

- **Pareto Optimality**: we call that an allocation is Pareto Optimal if no agent can increase his utility without worsening someone else. Formally, a partition $\{X_i\}_{i \in [n]}$ is Pareto optimal if there does not exists any other partition $\{Y_i\}_{i \in [n]}$ such that for all i $u_i(Y_i) \geq u_i(X_i)$ and there exists an agent i s.t. $u_i(Y_i) > u_i(X_i)$.
- **Envy-freeness (EF)**: we call an allocation $\{X_i\}_{i \in [n]}$ envy free if all agents prefer their items before anyone else’s items, formally, for all $i, j \in [n]$ we have that $u_i(X_i) \geq u_i(X_j)$. But this property can not always be satisfied, just take two agents and a single good valued positively by both agents.
- **EF1**: an easier property derived from the former is what we call Envy-freeness except for one good. Each agent should prefer his items to everyone else except for maybe one item from the bunch of another agent. Formally, we need that for all $i, j \in [n]$ there exists an $I \in X_j$ such that $u_i(X_i) \geq u_i(X_j \setminus \{I\})$.

The other question is how to derive a general utility that “summarizes” the utilities from all agents (we call this function a **social welfare function**). There are also several possible answers. Usually we want that those answers have some desirable properties. The main ones would be (Moulin [2003]):

- **Monotonicity**: if someone increases his utility without affecting the others utility the social welfare function should not decrease. This is basically the same as requiring Pareto Optimality for the optimum allocation.
- **Symmetry**: every permutation of an allocation should have the same social welfare. This assumes that all the agents are “equally important”. There is some work in the generalized

setting where this does not hold (Chakraborty *et al.* [2022]) but we will not discuss this generalized setting.

- **Continuity:** the function has to be continuous in the allocated utilities of each agent. This is if we think $F : \mathbb{R}^n \rightarrow \mathbb{R}$ where n is the number of agents and the inputs are the total utility received by each agent, the function is continuous in each variable.
- **Weak invariance up to scaling:** if we multiply all agents utility by the same factor, the optimal allocations should remain optimal. This says that we do not care about measuring the welfare in dollars, cents of a dollar or million dollars.

We discuss 3 possible social welfare functions:

The first obvious objective could be to maximize the sum of the utilities of the agents. This focuses on “efficiency”, we will allocate each item to the agent that values it the most. We have then that $F_{ef}(\{X_i\}) := \frac{\sum u_i(X_i)}{n}$. This function could not be desirable from an egalitarian point of view: all agents might be allocated to the same single agent causing that the rest of the agents get nothing.

In the other side of the spectrum we have the egalitarian measure: the objective is that the smallest utility is the biggest it can be. This is usually known as the max-min objective with $F_{eq}(\{X_i\}) := \min u_i(X_i)$. This kind of approaches that fall under the name of Santa Claus problems (Bansal and Sviridenko [2006]) lack from the efficiency point of view, we might allocate items to agents that value them much less than others in order to ensure equality.

In the “middle” of the two previous utility functions we have the Nash Social Welfare function: the geometric mean of the utilities. We define $F_{NSW}(\{X_i\}) := (\prod u_i(X_i))^{1/n}$. This function has some good qualities that made it to be widely studied (Caragiannis *et al.* [2019]; Kaneko and Nakamura [1979]). If we recall the means inequalities we have that $F_{eq} \leq F_{NSW} \leq F_{ef}$ with equality if and only if all agents have the same utility. This means that this function makes a compromise between equality and efficiency. It increases when the utility of an agent increases provided all utilities are non-zero (as happens in F_{ef} but not always in F_{eq}) and it also increases if the inequality between two agents is reduced without reducing the total sum of utilities.

It is also “invariant up to scaling” (an stronger assumption than the already discussed “weak invariance up to scaling”). This means that if we multiply all the valuations of a single agent by a constant factor the optimal allocations will not change. This is convenient as we do not need that the agents use the same scale when giving utilities to the set of items. Finally it is *EF1* and Pareto compatible, in the sense that any allocation that maximizes *NSW* will be both *EF1* and Pareto optimal.

These good qualities have made that a good amount of work studies allocation problems under *NSW* maximization and it will also be the objective of our work.

1.2 Online problems

The problem of allocating goods among a group of agents to maximize the NSW is an interesting topic. However, in recent times, a lot of algorithms research has shifted to a more realistic and practical scenario: the *online* problem (Alon *et al.* [2006]; Buchbinder and Naor [2009]; Karp *et al.* [1990]; Mehta and others [2013]; Devanur and Jain [2012]). In this setting, items arrive continuously over time and their values are only revealed when they arrive. The allocation of these items must be done immediately once their value is revealed. This scenario better represents the frequent, web-based procedures that occur daily.

For instance, the distribution of computational resources in a high-performance computer to researchers at a university. Researchers may request resources at different times throughout the day, and these requests must be handled as they arrive. The goal is to prioritize researchers with upcoming deadlines while also not hindering other users. This setting adds complexity to the problem, as we do not know the values of the items that will later come. Then, a decision that looks good in the present could be bad if we knew all future items.

In contrast we will talk about *offline* problems when the input is known in advance, i.e. what everyone understands as a regular algorithmic problem.

Online problems have been studied in very different settings. When we study offline algorithms we are usually interested in the *efficiency* of the algorithms as the number of instructions that it executes, as any Computer Science student knows. In the online setting the efficiency takes a secondary role and the algorithms are usually compared by their *competitive ratio* (CR). Given an algorithm ALG , an input X and a function to maximize (minimize) F we are interested in $CR_{\text{ALG}}^X := \frac{F_{\text{OPT}}(X)}{F_{\text{ALG}}(X)}$ ($CR_{\text{ALG}}^X := \frac{F_{\text{ALG}}(X)}{F_{\text{OPT}}(X)}$) where $F_{\text{OPT}}(X)$ is the optimum of F given X in an offline manner and $F_{\text{ALG}}(X)$ is the value obtained by the algorithm (or its expectancy if we are dealing with randomized algorithms). Given a set of inputs I (usually all possible inputs) we define $CR_{\text{ALG}} := \sup_{X \in I} CR_{\text{ALG}}^X$. Note that an algorithm with $CR = 1$ would be the best possible scenario, and the greater the competitive ratio the worst.

1.3 Randomized algorithms

Randomized algorithms is a wide class of algorithms that use some sort of randomness, usually given by a mechanism that can generate random bits. Those algorithms have shown to be basic for a lot of problems (Karp [1991]). They can be used for speeding up calculations, for obtaining approximated algorithms in intractable problems as the well-known 7/8 approximation of 3-SAT, for simplicity (Motwani and Raghavan [2018])...

But when facing online problems they are useful for another reason: they can “hedge risk”. Let’s imagine a setting where we have two agents and two items, the first item is valued 1 by both agents but the second is valued by 1 by one of the agents and 0 by the other. Any deterministic algorithm will get a NSW of 0 in the input where the agent that it allocates the first item is the one that positively values the second item. In the other hand an algorithm that chooses who to

allocate the first item randomly has an expected NSW of $1/2$.

From this we can see that random algorithms will play a central role in our work. We can summarize the explained about online problems and random algorithms with the following classical problem.

1.3.1 Ski-rental problem

This is a classic problem that shows the strength of random algorithms in online problems. Suppose that we will go skiing for a number of days N that we do not know before hand. Each day that we go skiing we can choose to either rent skis for 1 dollar or buy them for $C > 1$ dollars (we will suppose $C \in \mathbb{Z}$), in which case we will not need to rent them in any future day. We want to minimize the money spent. This is, if N was known before hand we would spend $\min(N, C)$, renting every day if $N < C$ and buying in the other case.

We can analyze what happens with deterministic algorithms. Suppose that a deterministic algorithm ALG buys the skis in the day T (that might depend on C) and let C_{ALG} be the cost of the algorithm and $C_{OPT} = \min(N, C)$ the optimal cost. We have that $C_{ALG} = N$ if $N < T$ and $C_{ALG} = T - 1 + C$ otherwise. Then we have 3 possibilities for the competitive ratio. Note that when $N \geq T$ taking $N = T$ will be the worst thing for the algorithm as it will not increase the cost from there.

- $N \leq C$ and $N < T$: $CR = \frac{N}{N} = 1$.
- $N \leq C$ and $N \geq T$ ($N = T$): $CR = \frac{T-1+C}{N} = \frac{T-1+C}{T}$.
- $N > C$ and $N < T$: $CR = \frac{N}{C}$
- $N > C$ and $N \geq T$ ($N = T$): $CR = \frac{T-1+C}{C} = \frac{N-1+C}{C}$. Note that this CR is always greater that the previous one.

Then we have that the competitive ratio will happen when $N = T$ and will be either $\frac{T-1+C}{T}$ or $\frac{T-1+C}{C}$. We want to take a T that minimizes the maximum of the two previous ratios. This will happen when $\frac{T-1+C}{T} = \frac{T-1+C}{C} \iff C = T$ and we have that $CR = \frac{2C-1}{C} = 2 - 1/C$. Then the optimal algorithm buys the skis in the C -th day and taking C arbitrarily large has a competitive ratio of 2.

We can use randomness to improve this algorithm designing a distribution over the number of days until buying the skis, and picking a number of days randomly following the distribution. This reduces the competitive ratio as for every number of days N we will have some probability of making a good decision. We can see how this process work in the following lemma:

Lemma 1. *There exists a random algorithm with competitive ratio $\frac{1}{1-(1-1/C)^C}$ which is at most $\frac{e}{e-1} \approx 1.58$.*

Proof. Let $D = \{p_i\}_{i \in \mathbb{Z}^+}$ the distribution where p_i is the probability that we buy the skis in the i -th day. Then if we have N days we will have an expected cost of: $A_N^D = \sum_{i \leq N} (i-1+C)p_i + N \sum_{i > N} p_i$.

We want to find the infimum c such there exists some D with $\frac{A_N^D}{\min(C, N)} \leq c$, this is the competitive ratio is at most c for any number of days.

Note that when $N \rightarrow \infty$ we have $\frac{\sum_{i \in \mathbb{Z}^+} (i-1+C)p_i}{C} \leq c$. We have that $\sum_{i \in \mathbb{Z}^+} (i-1+C)p_i > \sum_{i \leq N} (i-1+C)p_i + N \sum_{i > N} p_i$ for all $N \geq C$. Then we can delete all conditions with $N \in [C, \infty)$ and keep conditions with $N \in [C-1] \cup \{\infty\}$. Note now that for this set of conditions the cost of the strategy with $i = C$ (buying the day C) day is smaller than the cost of buying on any day with $i > C$. Then we can assume that $p_i = 0$ for all $i > C$. We are left with a finite system of inequalities: $I_N : \sum_{i \leq N} (i-1+C)p_i + N \sum_{N < i \leq C} p_i \leq cN$ for $N \in [C-1]$ and $I_C : \sum_{i \leq C} (i-1+C)p_i \leq cC$. We want to see that we can substitute the inequalities by equalities. Suppose that c^* is the minimum possible c (we can take it as a minimum by compactness) and an inequality is slack, let it be I_a . Then note suppose we take $p'_a = p_a + \epsilon, p'_{a+1} = p_{a+1} - \epsilon$ for an small enough $\epsilon > 0$. This does not affect the previous inequalities and creates an slack in inequality I_C . Then we can suppose than inequality C is slack. Taking $p'_1 = p_1 - \epsilon, p'_C = p_C + \epsilon$ we have that every inequality is slack, a contradiction.

Calling E_i to the corresponding equalities take $E'_i := E_i - E_{i-1}$ for $a > 1$ and $E'_1 = E_1$ we have $E'_i : Cp_i + \sum_{j > i} p_j = c$. Solving the now triangular system we get $p_i = \left(\frac{C-1}{C}\right)^{C-i} \frac{c}{C}$. Recalling now that $\sum p_i = 1$ we have $c = \frac{1}{1-(1-1/C)^C} < \frac{e}{e-1}$, reaching the bound in the limit when $C \rightarrow \infty$. A step by step when $C = 4$ is at figure 1.3.1. ▲

We have been able to design a distribution that defines a random algorithm, “hedging” the decisions and improving the deterministic algorithm lower bound. We will use this same idea in Chapter 3.

1.3.2 Yao’s Principle

From the previous example we can see how to interpret a randomized algorithm as a distribution over deterministic algorithms. This can help into stating a principle that helps us to establish upper bounds on the utility of any randomized algorithm. This is known as Yao’s principle:

Theorem 1 (Yao’s Principle). *Consider a set of inputs \mathcal{X} and let \mathcal{A} be the set of all deterministic algorithms that solve the problem. Let $u : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}^{\geq 0}$ be the utility function. For any probability distribution q over \mathcal{X} and any randomized algorithm with distribution p over \mathcal{A} we have:*

$$\min_{x \in \mathcal{X}} \mathbb{E}_p[u(x, A)] \leq \max_{a \in \mathcal{A}} \mathbb{E}_q[u(X, a)]$$

This is the worst case of any randomized algorithm is at most the expected value of the best deterministic algorithm over the distribution of inputs.

Proof.

$$\min_{x \in \mathcal{X}} \mathbb{E}_p[u(x, A)] = \sum_{y \in \mathcal{X}} q_y \min_{x \in \mathcal{X}} \mathbb{E}_p[u(x, A)] \leq \sum_{y \in \mathcal{X}} q_y \mathbb{E}_p[u(y, A)] = \sum_{y \in \mathcal{X}} \sum_{b \in \mathcal{A}} p_b q_y u(y, b) =$$

$$\begin{array}{l} \text{inf } c \text{ s.t.} \\ \left\{ \begin{array}{ll} Cp_1 + p_2 + p_3 + p_4 + p_5 + \cdots & \leq c \cdot 1 \\ Cp_1 + (C+1)p_2 + 2p_3 + 2p_4 + 2p_5 + \cdots & \leq c \cdot 2 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + 3p_4 + 3p_5 + \cdots & \leq c \cdot 3 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + (C+3)p_4 + 4p_5 + \cdots & \leq c \cdot 4 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + (C+3)p_4 + (C+4)p_5 + \cdots & \leq c \cdot 4 \\ \vdots & \leq c \cdot 4 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + (C+3)p_4 + (C+4)p_5 + \cdots & \leq c \cdot 4 = cC \end{array} \right. \end{array} \quad \rightarrow$$

$$\begin{array}{l} \text{inf } c \text{ s.t.} \\ \left\{ \begin{array}{ll} Cp_1 + p_2 + p_3 + p_4 + p_5 + \cdots & \leq c \cdot 1 \\ Cp_1 + (C+1)p_2 + 2p_3 + 2p_4 + 2p_5 + \cdots & \leq c \cdot 2 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + 3p_4 + 3p_5 + \cdots & \leq c \cdot 3 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + (C+3)p_4 + (C+4)p_5 + \cdots & \leq c \cdot 4 = cC \end{array} \right. \end{array} \quad \rightarrow$$

$$\begin{array}{l} \text{min } c \text{ s.t.} \\ \left\{ \begin{array}{ll} Cp_1 + p_2 + p_3 + p_4 & \leq c \cdot 1 \\ Cp_1 + (C+1)p_2 + 2p_3 + 2p_4 & \leq c \cdot 2 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + 3p_4 & \leq c \cdot 3 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + (C+3)p_4 & \leq c \cdot 4 = cC \end{array} \right. \end{array} \quad \rightarrow$$

$$\begin{array}{l} \text{min } c \text{ s.t.} \\ \left\{ \begin{array}{ll} Cp_1 + p_2 + p_3 + p_4 & = c \cdot 1 \\ Cp_1 + (C+1)p_2 + 2p_3 + 2p_4 & = c \cdot 2 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + 3p_4 & = c \cdot 3 \\ Cp_1 + (C+1)p_2 + (C+2)p_3 + (C+3)p_4 & = c \cdot 4 = cC \end{array} \right. \end{array} \quad \rightarrow$$

Figure 1.1: Steps of the proof of Lemma 1 with $C = 4$ (although a general C is used when possible)

$$\sum_{b \in \mathcal{A}} p_b \mathbb{E}_q[u(X, b)] \leq \sum_{b \in \mathcal{A}} p_b \max_{a \in \mathcal{A}} \mathbb{E}_q[u(X, a)] = \max_{a \in \mathcal{A}} \mathbb{E}_q[u(X, a)]$$



1.4 A proper definition of our problem

Consider n agents and T indivisible items, where each item $t \in [T]$ arrives in a sequential manner. Each agent has value $v_i(\{t\}) \geq 0$ for the item t that is revealed to us only at the time of its arrival. Let X_i^t be the set of items allocated to agent i as of round t , and let $X_i = \bigcup_t X_i^t$. Thus, the total value of items allocated to an agent i is given by $v_i(X_i) = \sum_{x \in X_i} v_i(x)$. Our goal is to produce an allocation $X = \{X_1, \dots, X_n\}$ of the set $[T]$ that maximizes the value of the *Nash social welfare* (NSW).

Definition 1 (Nash Social Welfare). *The **Nash Social Welfare** (NSW) of an allocation X is defined as the geometric mean of agents' valuations:*

$$NSW(X) = \left(\prod_i v_i(X_i) \right)^{1/n}.$$

In case we use a randomized algorithm which possibly induces multiple allocations given the same sequence of items, we take expectation on the NSW and abuse it by NSW of the algorithm.

We here measure the quality of an online algorithm in terms of their *competitive ratio* with respect to the above objective. Let $NSW_{\text{ALG}}(\mathcal{I})$ denote the NSW value of the allocation induced by an online algorithm on a given instance \mathcal{I} , and let $NSW_{\text{OPT}}(\mathcal{I})$ denote the optimal such objective value. Then the competitive ratio (CR) of the objective is defined as

$$\alpha = \sup_{\mathcal{I}} \frac{NSW_{\text{OPT}}(\mathcal{I})}{NSW_{\text{ALG}}(\mathcal{I})}.$$

We often say that ALG is α -approximate if it has a competitive ratio of α .

Until now we have not defined properly how the items arrive in our online problem. This is an important property that can affect greatly in the difficulty of the problem and thus in the competitive ratio. We will consider the three following models, from stronger to weaker:

Definition 2 (Adaptative Online Input). *Just before every item arrives an adversary selects the value of such item for all n agents, thus this can depend on the previous decisions by the algorithm.*

Definition 3 (Adversarial Input). *An adversary selects the value of each arriving item t for all n agents, as well as the order in which these items arrive before the algorithm starts.*

Definition 4 (Random Order Input). *An adversary selects the value of each arriving item t for all n agents before the algorithm starts, but not the order. This means that we will be interested in the expected NSW for all the possible orderings of the selected input.*

The first input model is so strong that randomization does not help. Intuitively any hedging that we could do via randomization will be canceled as the adversary will pick the worst possibility after the random decision has been made. Thus we will have to pick the decision that maximizes the worst outcome, that is exactly what a deterministic algorithm does. This can be formalized as found in Ben-David *et al.* [1990]:

Theorem 2. *If there is a randomized algorithm that is α -approximate against any adaptive online adversary then there also exists an α -approximate deterministic algorithm.*

We will usually work with the **Adversarial Input** model, but we will also discuss the two other models in some settings.

1.5 Related problems

In this section we discuss some similar problems that might hint us in which kind of results we will have in the work. We mainly discuss similar objective functions, the offline version of the problem and the divisible items case:

1.5.1 NSW offline maximization

The offline version of the *NSW* maximization problem has been a widely studied problem. The first works focused on the computational complexity. The work of Magnus and Jörg [2010] showed that the problem is **NP-Complete** via the *Partition* problem. Then Lee [2017] proved that the problem is also **APX**. The class of **APX-Hard** contains those problems that admit polynomial time algorithms that give a constant factor approximation. The survey of Nguyen *et al.* [2013] is a good reference in this kind of questions.

The first positive results is found on the work of Cole and Gkatzelis [2015] where they present both a weakly-polynomial and a strong-polynomial algorithm that find $2 \cdot e^{1/e}$ approximations. Interestingly they use the Eisenberg-Gale program (see next subsection) in their algorithms in order to hint how the integral allocations should work. They tightened their result to a factor of 2 in Cole *et al.* [2017].

The work of Barman *et al.* [2018] lowered the factor to $e^{1/e}$ that matches the integrality gap (the ratio between the integral and relaxed solutions) of the relaxation that Cole *et al.* [2017] stated. We strongly take advantage of their results in Chapter 3.

1.5.2 Divisible items online problem

We have seen that the offline problem with indivisible items is a difficult problem, both *NP*-hard and *APX*-hard. We can also consider the divisible setting. In this case given an item valued v_i by agent i we can split it between agents, giving each agent a $x_i \in [0, 1]$ fraction on the item and receiving a utility of $x_i v_i$, as far as $\sum x_i = 1$. In this case the problem can be stated as a convex

program, known as Eisenberg-Gale program, just by taking the logarithm of the objective function:

$$\begin{aligned} & \text{maximize} && \sum_{i=1}^n \log\left(\sum_{j=1}^m x_{ij} v_i(j)\right) \\ & \text{subject to} && \sum_{i \in [n]} x_{ij} = 1, && \forall j \in [m] \\ & && x_{ij} \geq 0, && \forall i \in [n], \forall j \in [m] \end{aligned}$$

Although this optimization problem can not be solved with the standard polynomial linear programming techniques (as it is not linear), [Devanur *et al.* \[2008\]](#) showed that it can be solved in polynomial time, although the question whether it can be solved in *strongly* polynomial time remains open. This fact hints us that the divisible setting is easier than the indivisible one, and this is the case in the online setting as well.

The work in the offline setting received two interesting contributions last year (2022). The work of [Banerjee *et al.* \[2022\]](#) provided an algorithm using *predictions*, this is an algorithm that is also given the sum of valuation of all items for each agent before receiving any items. They also showed that the algorithm is nearly asymptotically tight versus an adversarial online input.

Trying to avoid predictions the work [Huang *et al.* \[2022\]](#) studies a setting where the total valuations of the agents are balanced or that the optimal assignments are balanced. Under this setting they provide similar results of the ones provided by [Banerjee *et al.* \[2022\]](#).

Both works use two similar techniques that are impossible (or at least very difficult) to translate into the indivisible setting: first they allocate half of every item evenly so they make sure that no agent will receive a very low total utility. For the second half of every item they solve a variant of the Eisenberg-Gale program and allocate based on that. Although sometimes the fractional problems can be translated into randomized algorithms interpreting the fraction as a probability this is not the case for our problem as we will be able to show via our more restricting lower bounds. This implies that the ideas of the divisible setting are difficultly applied to our setting.

1.5.3 Online Santa Claus problem

In the previous sections we discussed the sum of utilities and the *max-min* function as interesting objectives to maximize instead of the *NSW*. In the context of additive valuations maximizing the sum of utilities is trivial as it is enough to allocate each item to the agent that values it the most. Maximizing the *max-min* objective is a more interesting task. This problem is known as the Santa Claus problem and has been widely studied in both online and offline setting.

The online problem is similar to ours in some aspects: you have to ensure that every agent receives a good amount of utility, which is a problem specially when few items are available and some agents might have no allocated items at all. Note that given a divisible settings with equal valuation for all agents both quantities are maximized allocating the same utility for all agents. It also has to deal with a sudden arrival of big items that implies that having a very similar value for all agents might drawback. In the other hand for maximizing the *NSW* we do not only care about

Bound	Identical	Affine	Binary	Binary (many items)	m -Bivalue
Upper	1.4447 (Thm 5)	$1.4447 \cdot B$ (Thm 13)	–	$\Theta(n)$ (Thm 14)	m (Thm 17)
Lower	Deter: 1.4422 (Thm 6) Rand: 1.3692 (Thm 10)	Deter: ∞ (Thm 12) Rand: 1.3692 (Thm 10)	$e^{\Theta(n)}$ (Thm 3)	$\Theta(n)$ (Thm 15)	$m^{5/18}$ (Thm 16)

Table 1.1: Summary of results. **Deter** denotes the deterministic algorithm and **Rand** refers to the randomized algorithm. For the upper bound in the affine utility setting, B denotes the quantity defined in (4.1). The parameter m in the bivalue setting denotes that each item has a value in $\{1, m\}$ for $m > 1$. We do not explore the upper bound in the general binary setting as the lower bound turns out to be too pessimistic so the matching algorithm would not be interesting.

the minimum but all the distribution of utilities, that ultimately translate on a more complicated problem.

The work of Hajiaghayi *et al.* [2022] is a source of inspiration for the present work. They deal with the binary setting that we also discuss in Chapter 5 and they overcome the inherent lower bounds that we also find in chapters 2 and 5 by considering the setting where a large number of items arrive. They are able to provide a $(1 - \epsilon)$ approximated algorithm in the random order setting given that the optimum is $\geq \Omega(\log n / \epsilon^2)$ (i.e: it is possible to allocate a value of at least $\log n / \epsilon^2$ to every agent at the same time). They see that the result is tight in n and that a lower bound exist if the dependence of ϵ is $1/\epsilon$ instead of $1/\epsilon^2$. They also discuss an algorithm and lower bound for the adversarial input setting that we improve in section 5.2.1.

1.6 Overview of the results

In the following chapters we study the online NSW-maximization problem under a various number of settings. In the next Chapter 2 we see that under important restrictions our problem lacks an exponential (in the number of agents) competitive ratio lower bound.

This directs us to consider restricted scenarios. We discuss the scenario where all agents have the same valuation function in Chapter 3 deriving constant lower and upper bounds on the competitive ratio. We study deterministic and random algorithms and we study some fundamental bounds on the setting together with parameterization of the competitive ratio when the values of the items follow some real world-inspired assumptions.

We present a generalization of the previous setting with *affine valuations* in Chapter 4.

Finally we discuss the setting where the valuations can only take two possible values in Chapter 5. When those values are 0 and 1 (we call it binary setting) we derive tight asymptotic results. We also derive some bounds when the possible values are both non-zero and show an interesting connection with the online maximum matching problem.

An overview of the main results can be checked at Table 1.1.

2. General lower bounds

We will start showing that any algorithm lacks from an competitive ratio of at least $e^{\Theta(n)}$ where n is the number of agents. This holds for any input model and even if we have predictions on the total utility of the agents, and we restrict the valuations to be binary (a setting that is discussed more in-depth in chapter 5). This results shows how the indivisible setting is more complex than the divisible setting comparing this results with the paper of [Banerjee *et al.* \[2022\]](#) where they are able to get logarithmic-bounded competitive ratios when they are able to access predictions.

We are able to derive this bounds exploiting the fact that the NSW is 0 whenever the utility of an agent is 0. Imagine that the algorithm receives an item valued by 1 by two agents, and that one of those agents will not value positively any other item. With probability $1/2$ the algorithm will allocate the item to the “wrong” agent. Pairing $2n$ agents in this manner we would have a probability of getting all the guesses right of $(\frac{1}{2})^n$.

We formalize this idea in the next theorem. Note that we also restrict us to the easier input model (the random order one) and we give the algorithm predictions, as done in the aforementioned work of [Banerjee *et al.* \[2022\]](#).

Theorem 3. *In the binary value setting, for any (randomized) algorithm, the competitive ratio is at least $e^{\Theta(n)}$, even if the algorithm knows the total utility of agents beforehand and the items arrive in random order.*

Proof. For an integer $k \geq 1$, set $n = T = 3k$. We will specify a distribution over instances with n agents and T items and show that for any algorithm, the expected value of NSW is at most $(\frac{8}{9})^{n/3}$ times the optimal Nash social welfare. Here the expectation is over both the randomness of the algorithm *and* the randomness of the instance. This implies the theorem’s statement since it means the algorithm should satisfy the condition for at least some instance, by a probabilistic method argument.

We let x_1, \dots, x_T denote the items in question and note that, in the random order setting, g_1, \dots, g_T are a random permutation of x_1, \dots, x_T . We will specify each instance by the vector of valuations for each item x . We use the notation $\mathbf{v}(x) := (v_i(x))_{i \in [n]}$ to denote this vector. For each $j \in [0, k - 1]$, we independently draw an integer $\tau(j) \in \{3j + 1, 3j + 2, 3j + 3\}$. We then set the valuations of items $x_{3j}, x_{3j+1}, x_{3j+2}$ as

$$\begin{aligned} \mathbf{v}(x_{3j+1}) &:= e_{3j+1} + e_{3j+2} + e_{3j+3} \\ \mathbf{v}(x_{3j+2}) &:= e_{\tau(j)} \\ \mathbf{v}(x_{3j+3}) &:= e_{3j+1} + e_{3j+2} + e_{3j+3} - e_{\tau(j)}. \end{aligned}$$

In other words, two of the agents in $\{3j + 1, 3j + 2, 3j + 3\}$ have valuation 1 for x_{3j+1} and x_{3j+3}

while the other agent has valuation 1 for $x_{3j+1} x_{3j+2}$. Defining $\text{NSW}_j := (\prod_{i=3j+1}^{3j+3} v_i(X_i))^{1/n}$. We decompose $\text{NSW}(X)$ as

$$\text{NSW}(X) := \prod_{j=0}^{k-1} \text{NSW}_j.$$

The main intuition behind the proof is as follows: for each $j \in [0, k-1]$, we have $\text{NSW}(X) = 0$ with probability at least $1/9$. This is because if x_{3j+1} arrives first, then, with probability $1/3$, the algorithm will allocate it to the agent who values x_{3j+2} as well, which leads to $\text{NSW} = 0$ as at least one of the agents will not be satisfied. Given the independence between the NSW_j , this implies the desired bound of the theorem statement. Obtaining formal proof requires careful reasoning however given the random nature of the argument.

Let S denote the random seed of the algorithm. Let σ denote the underlying permutation of the elements. Assume that we further randomly permute σ_j for $j \in [0, k]$ to change the inner ordering of the elements $x_{3j+1}, x_{3j+2}, x_{3j+3}$. We need to show that

$$\mathbb{E}_{\sigma, (\sigma_j)_{j=0}^k, (\tau_j)_{j=0}^k, S} [\text{NSW}] < \left(\frac{8}{9}\right)^k.$$

Fix the value of σ, S . We will show that

$$\mathbb{E}_{\sigma_j, \tau_j} [\text{NSW} | \sigma, S] < \left(\frac{8}{9}\right)^k.$$

This implies the statement by the law of iterated expectation. For the rest of the proof, we assume the conditioning on σ, S and will not explicitly specify it in the notation.

Sort the integers $j \in [0, k-1]$ in the order of first appearance of some $x \in \{x_{3j+1}, x_{3j+2}, x_{3j+3}\}$ and let σ' be the corresponding permutation. Formally, we assume that the first element in $\{x_{3\sigma'(j-1)+1}, x_{3\sigma'(j-1)+2}, x_{3\sigma'(j-1)+3}\}$ arrives before the first element in $\{x_{3\sigma'(j)+1}, x_{3\sigma'(j)+2}, x_{3\sigma'(j)+3}\}$. We note that σ' is fully determined by σ , and is therefore fixed given the conditioning on σ , as it is not affected by τ_j or σ_j .

We first make the following two claims.

Claim 1. *The value of $\text{NSW}_{\sigma'(j)}$ is fully determined by $\text{BEF}_j := \{(\sigma_{\sigma'(j')}, \tau_{\sigma'(j')}) : j' < j\}$.*

Proof. We first observe that the algorithm's allocation is fully determined by the items it has previously seen and its internal randomness. The past items are fully determined however by BEF_j and the internal randomness is also fixed since we have conditioned on S . \blacktriangle

Claim 2. *For each $j \in [0, k-1]$, conditioned on BEF_j , we have $\text{NSW}_j = 0$ with probability at least $1/9$.*

Proof. Since τ_j, σ_j were assumed to be i.i.d, with probability at least $1/3$, $x_{3\sigma'(j)+1}$ arrives before $x_{3\sigma'(j)+2}$ and $x_{3\sigma'(j)+3}$. When the algorithm sees $x_{3\sigma'(j)+1}$, it needs to irrevocably decide which

agent to allocate it to. The decision is deterministic at this point however since we have fixed on the internal randomness of the algorithm, the items that have already arrived (by conditioning on BEF_j) and the current item (by assuming that the algorithm sees $x_{3\sigma'(j)+1}$). Assume that it allocates the item to x_1 (the other cases are handled similarly by symmetry). With probability $\frac{1}{3}$, we have $\tau(j) = 1$. It is clear that at least one of $3j + 2$ and $3j + 3$ will have utility zero because x_{3j+3} can only be allocated to one of them. Therefore, NSW becomes zero. The overall probability of this happening is at least $\frac{1}{3} \cdot \frac{1}{3} = \frac{1}{9}$, finishing the proof. \blacktriangle

We now use the following lemma.

Lemma 2. *Let X_1, \dots, X_k and Y_1, \dots, Y_k be random variables such that for each $i \in [k]$:*

1. X_i is fully determined by Y_1, \dots, Y_i and
2. $\mathbb{E}[X_i | Y_1, \dots, Y_{i-1}] \leq c$ for some fixed c .

Then $\mathbb{E}[\prod X_i] \leq c^k$.

Proof of the Lemma. We prove by reverse induction that

$$\mathbb{E} \left[\prod_{j \geq i} X_j | Y_1, \dots, Y_{i-1} \right] \leq c^{k-i}.$$

Setting $i = 0$ then finishes the proof.

For $i = k$, the claim holds trivially. Assuming it holds for $i + 1$, we will show it holds for i as well. Fix the value of Y_1, \dots, Y_{i-1} . Letting the abbreviation $Y_{\leq i}$ denote Y_1, \dots, Y_i ,

$$\begin{aligned} \mathbb{E} \left[\prod_{j \geq i} X_j | Y_{\leq i-1} \right] &= \mathbb{E}_{Y_i} \left[\mathbb{E} \left[\prod_{j \geq i} X_j | Y_{\leq i} \right] \right] \\ &= \mathbb{E}_{Y_i} \left[X_i \mathbb{E} \left[\prod_{j \geq i+1} X_j | Y_{\leq i} \right] \right] \\ &\leq \mathbb{E}_{Y_i} \left[X_i c^{k-i+1} \right] \\ &\leq c^{k-i}, \end{aligned}$$

where the first equality follows from the law of iterated expectation, the second equality follows from the fact that X_i is fully determined by $Y_{\leq i}$ (recall that we have fixed $Y_{\leq i-1}$), the first inequality follows from the induction hypothesis and the final inequality follows from the assumption $\mathbb{E}[X_i | Y_{\leq i}] \leq c$. \blacktriangle

Invoking Lemma 2 (with X_j set to $\text{NSW}_{\sigma'(j)}$ and Y_i set to $(\sigma_{\sigma'(j)}, \tau_{\sigma'(j)})$) finishes the proof. We note that the required conditions of the lemma are satisfied because of Claims 1 and 2. \blacktriangle

Note that we get a competitive ratio of $(\frac{9}{8})^n$ but that the base of the exponential is not optimal. Dropping the predictions assumption it is easy to derive stronger results but we do not think that it is very interesting as the exponential nature of the results.

Changing the input model from random order to adversarial input model we can derive a stronger competitive ratio of $n!$ using a very similar idea:

Theorem 4. *Under an adversarial input, the competitive ratio for n agents is at least $n!$.*

Proof. Consider the set of valuations for n agents and n items $S = \{ \{e_{\sigma(1)} + e_{\sigma(2)} + \dots + e_{\sigma(n)}, e_{\sigma(2)} + \dots + e_{\sigma(n)}, \dots, e_{\sigma(n-1)} + e_{\sigma(n)}, e_{\sigma(n)} \} | \sigma \text{ permutation of } n \text{ elements} \}$ and the uniform probability q in S . The optimal allocation will set allocate the i -th item to the $\sigma(i)$ -th agent getting a NSW of 1. Now every deterministic algorithm will get a NSW of 1 in only one of the inputs in S and 0 in the rest. Then by Yao's principle the expected NSW of any random algorithm is bounded above by $1/n!$. The stated competitive ratio follows. ▲

Note than any deterministic algorithm will get a competitive ratio of ∞ so it will be also be the competitive ratio of the online adversarial input.

This results discourages us to talk about positive results as they will be exponentially bad. This is also the reason that in the following sections we study other settings with extra assumptions about the utility function.

3. Equal valuations setting

After the negative results in the general settings we focus on a simplification of the problem. Here we suppose that all agents value items in the same way. This is for all $i, j \in [n]$ we have $v_i = v_j$.

This setting is the simplest when we think about real-world problems. Here we do not have items that valued differently by agents, but a same utility that is agreed for everyone. This can be the case of well established products that have a price that is widely accepted. Take for example an university that received some new computers with different specifications and wants to divide them to different faculties such that the received value in dollars for each faculty is well distributed under the Nash Social Welfare point of view.

One interesting thing is that in the case when $n = 2$ this reduces to the partition problem. In this problem we are given a set of non-negative numbers and we want to make two partitions such that each one that adds up to as near to half of the sum as possible. This is known to be a NP-problem. We can easily reduce our problem to it noticing that we want to maximize $\sqrt{a_1 a_2} = \sqrt{a_1(S - a_2)}$ where a_i is the value allocated to the i -th agent and S is the total sum. This expression is maximized when $a_1 = \frac{S}{2}$.

We will start the chapter showing that the greedy algorithm achieves a bounded competitive ratio. Through this we will see the connection of NSW with $EF1$ and we will use a result from Barman *et al.* [2018]. Then we will closely examine the difficulties encountered by any deterministic algorithm when facing with this problem and will study the connection of this problem with the function $x^{1/x}$. We will be able to find lower bounds on the performance of any randomized algorithms using the previously discussed. Finally we propose a randomized algorithm to break the lower bound of the deterministic algorithms.

3.1 Greedy algorithm

Algorithm 1: Greedy

```
1 Given any deterministic tie-breaking rule
2 Initialize  $X_1, \dots, X_n = \emptyset$ 
3 for  $t = 1, 2, \dots, T$  do
4   | Item  $g_t$  arrives ;
5   | Find the least satisfied agent  $j = \operatorname{argmin}_{i \in [n]} v_i(X_i)$ ;
6   |  $X_j = X_j \cup \{g_t\}$  ;
7 end
```

The greedy algorithm 1 looks as a very naive algorithm. We will show that it has a tight competitive ratio of $e^{1/e}$. Not only this but it is easy to show that it is $EF1$, a desirable property.

We will in fact show that any *EF1* has a competitive ratio of at least $e^{1/e}$ so the greedy is optimal in this class of algorithms.

We begin seeing the aforementioned *EF1* property:

Lemma 3. *At each stage of the greedy procedure of Algorithm 1, the allocation is EF1.*

Proof. We prove the result by induction on the arriving items t . For $t \leq n$, each agent has been allocated at most one item and is thus trivially *EF1* by removal of this item. Now, for $t > n$, assume that the allocation up to round $t - 1$ is *EF1* and, towards contradiction, that at round t it is not *EF1*. Since the allocation is not *EF1*, there exists two agents $i, j \in [n]$ such that

$$v(X_i^t) < v(X_j^t \setminus g) \text{ for any } g \in X_j^t.$$

If at round t , the arriving item, g_t , was not assigned to this envied agent, then the allocation was not *EF1* at $t - 1$. Thus, we assume g_t is assigned to j and in the previous round, we had:

$$v(X_i^{t-1}) \geq v(X_j^{t-1}) = v(X_j^{t-1}).$$

This further implies that after allocation the t -th item,

$$v(X_i^t) \geq v(X_j^t \setminus g_t)$$

which implies the allocation is *EF1*. Contradiction. ▲

This property will imply the $e^{1/e}$ upper bound on the competitive ratio as shown by [Barman et al. \[2018\]](#). We will define ϵ -*EF1* for citing the result properly:

Definition 5 (ϵ -Envy-Free up to One Item (ϵ -*EF1*)). *Given any $\epsilon > 0$, an allocation is said to be ϵ -approximately envy-free up to one good (ϵ -**EF1**) if for every pair of agents $i, j \in [n]$, there exists a good $g \in X_j$ such that $(1 + \epsilon)v_i(X_i) \geq v_i(X_j \setminus g)$.*

Based on this definition, we recall the following lemma:

Lemma 4. *From [Barman et al. \[2018\]](#) under identical valuations, any ϵ -*EF1* allocation provides a $e^{(1+\epsilon)/e}$ -approximation to the offline optimal NSW.*

We now obtain our main upper bound approximate result.

Lemma 5. *At each stage of the allocation procedure, Algorithm 1 maintains an allocation that is at most $e^{1/e}$ -approximate.*

Proof. As proven in Lemma 3, Algorithm 1 is *EF1*. Thus, by application of Lemma 4 with $\epsilon \rightarrow 0$ we have the result. ▲

Once we have seen the upper bound we can see that it is matched by a lower bound that also applies to all *EF1* algorithms. The idea behind the proof of the following lemma is that by always

maintaining a very similar utility for all agents the algorithm can not react properly when a bunch of high-valued items arrive. We will see that this kind of problems arise not only on the *EF1* algorithms but in all deterministic algorithms in the following section.

Lemma 6. *For any $\epsilon > 0$, there exists an adversarial online input stream such that Algorithm 1 (and any algorithm that maintains the *EF1* property) returns an allocation that is at least $(e^{1/e} - \epsilon)$ -approximate.*

Proof. We first define a rational approximation of the universal constant e : let $p, q \in \mathbb{Z}$ be coprime such that $|p/q - e| \leq \delta$ for $\delta > 0$. We now consider a problem instance with p agents and $pq + (p - q)$ arriving items to be allocated whose value at time t is given by

$$v(g_t) = \begin{cases} \frac{1}{q} & \text{for } t \leq pq \\ c & \text{else} \end{cases}$$

for some constant $c > 0$. Consider c large enough to singularly satisfy an agent (ie. the optimal offline solution allocates items with value c as a singleton). Thus, $p - q$ agents receive such an item and have value c for their allocation bundle.

In the offline optimal allocation, the first pq items of value $1/q$ will be distributed uniformly across the remaining q agents. Thus, the optimal Nash social welfare for this instance is given by

$$\text{NSW}_{\text{OPT}} = \left(c^{(p-q)} \left(\frac{p}{q} \right)^q \right)^{1/p}$$

We now proceed by computing the Nash social welfare of the allocation Algorithm 1 yields. Most crucially, since the algorithm has no information on the later arriving “large enough” items, it must uniformly distribute the first pq items across the agents.

Claim 3. *Any *EF1* allocation must uniformly distribute the first pq items across the agents.*

Proof. Suppose not. After the first $t = pq$ rounds, by the given item values, we must have

$$\sum_{i \in [p]} v(X_i^t) = pq$$

and by assumption, there exists agents $i \neq j$ such that $|X_i^t| \neq |X_j^t|$. If the two bundles differ by more than two, then the allocation cannot be *EF1* – so we proceed in assuming $|X_i^t| - |X_j^t| \leq 1$ for any $i, j \in [p]$. Thus, at the given round, we must have the invariant

$$\left(\min_i |X_i^t| \right) x_1 + \left(\max_j |X_j^t| \right) x_2 = pq$$

where $x_1 + x_2 = p$ segregates the agents into those with smaller versus the larger bundle sizes. The

invariant can be rewritten as

$$\begin{aligned} & \left(\min_i |X_i^t| \right) x_1 + \left(\min_i |X_i^t| + 1 \right) x_2 = pq \\ & \iff (x_1 + x_2) \min_i |X_i^t| + x_2 = pq \\ & \iff x_2 \equiv 0 \pmod{p} \end{aligned}$$

a contradiction since $x_2 < p$. Thus, all allocation bundles must have equal size at round pq . \blacktriangle

The above claim verifies that Algorithm 1 must uniformly distribute the first pq items of smaller value across the agents. We can further see that the following $p - q$ items of large value must be distributed evenly to ensure an EF1 allocation. Therefore, the Nash social welfare of the allocation induced by the algorithm will be

$$\text{NSW}_{\text{ALG}} = ((1 + c)^{p-q})^{1/p}.$$

We can now compute the competitive ratio as compared to the optimum to be

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} = \left(1 + \frac{1}{c} \right)^{-(1-q/p)} \left(\frac{p}{q} \right)^{q/p} \rightarrow \left(\frac{p}{q} \right)^{q/p} \approx e^{1/e},$$

where we take $c \rightarrow \infty$ for getting an arbitrarily close to $e^{1/e}$ ratio. \blacktriangle

We can summarize the results of this section in the following theorem:

Theorem 5. *In the identical value setting and online adversarial input, greedy Algorithm 1 achieves the competitive ratio of $e^{1/e} \approx 1.4447$.*

Proof. Follows from Lemmas 5 and 6. \blacktriangle

3.2 Lower bound on deterministic algorithm

We will now derive a lower bound on any deterministic algorithm. The input that will prove this bound is similar to the one used for the greedy algorithm in the sense that it exploits the algorithm allocating “too evenly” some items. Since we are not restricted to EF1 algorithms this “too evenly” will be more loose and will give a ratio slightly smaller.

We present a lemma before the lower bound proof.

Lemma 7. *Given an integer n , let $P = (a_1, \dots, a_s) \in \mathbb{N}^s$ partition the integer into a any number of chunks s such that $\sum_{i \in [s]} a_i = n$. Consider $M_P = \prod_{i \in [n]} a_i$ and let M_n denote the maximum value of M_P through all possible partitions of $n = 3k + r, 0 \leq r \leq 2$ and S_n the number of chunks in such partition. Then:*

1. $M_n = 3^k$ for $r = 0$

2. $M_n = 4 \cdot 3^{k-1}$ for $r = 1$
3. $M_n = 2 \cdot 3^k$ for $r = 2$

Proof. Suppose that we have an optimal partition $x_1 \geq \dots \geq x_s \geq 0$ that maximizes the product and has the lowest number of $x_i = 4$ through all partitions that maximize the product. If $x_i \geq 4$ for some i , we can divide x_i into 2 and $x_i - 2$ and we have $2(x_i - 2) \geq x_i$ which contradicts the optimality.

If $x_i = 1$ for some i , then removing x_i and constructing $x'_j = x_i + 1$ increases the product. Therefore, the optimal allocation should consist of only size 2 and 3 chunks. Since $2^3 < 3^2$ and $2 \cdot 3 = 3 \cdot 2$, the optimal allocation should have as smaller number of 2 sized chunks as possible.

Hence in case of $n = 3k$, we have optimal partition of $(3, 3, \dots, 3)$. For $n = 3k + 1$, we can divide n into $n = 3k + 1 = 2 \cdot 2 + 3 \cdot (k - 1)$ and for $n = 3k + 2$, we have $n = 3k + 2 = 1 \cdot 2 + 3 \cdot k$. This completes the proof. \blacktriangle

Remark 1. Here we see the first connection with the function $x^{1/x}$. Let's consider a relaxation of the previous problem where the a_i are in \mathbb{R} instead of in \mathbb{N} . Then if we use s chunks the optimal partition will have $a_i = \frac{n}{s}$, which follows from AM-GM inequality. Hence we are left to maximize $(\frac{n}{s})^s$. If we drop the assumption that s is an integer we can rewrite the function as $(\frac{n}{s})^s = x^{n/x} = (x^{1/x})^n$ where $x := \frac{n}{s}$. This maximum is reached at $x = e$ obtaining $e^{n/e}$, that is a good approximation of M_n but with a greater value due the relaxation. Note that this matches the lower bound seen in the previous section.

With the previous lemma we can prove the result of this section, a lower bound on any deterministic algorithm.

Theorem 6. Given n agents, in the identical value setting, the competitive ratio of any deterministic algorithm is lower bounded by $(M_n)^{1/n}$, which is at most $3^{1/3} \approx 1.4422$, that is also the limit when $n \rightarrow \infty$ and the exact value for all $n = 3k$.

Proof. Consider the adversarial construction where $m = k + n$ where k is set so $n - k = S_n$. The adversary selects the value of the arriving goods to be

$$v_t = \begin{cases} 1 & \text{for } 0 < t \leq n \\ M & \text{for } n < t \leq n + k \end{cases}$$

where M is sufficiently large. The optimal allocation, denoted as OPT, will allocate the large items as singletons and distribute the remaining 1 valued items across the remaining agents. This distribution will be exactly the one of Lemma 7 that yields:

$$\text{NSW}(\text{OPT}) \geq \left(M^k M_n\right)^{1/n}.$$

To upperbound the objective value of any online algorithm, we prove the following claim:

Claim 4. *Any online algorithm with a CR $< \infty$ will allocate the first n items to the n different agents.*

Proof. For any number of items m suppose consider a sequence of items with valuations $\{v_i\}$ where $v_i = 0$ for $i > n$ and $v_i > 0$ for $i \leq n$. If the first n items are not allocated evenly the competitive ratio is ∞ . \blacktriangle

From the previous claim we know that at the end of the first n items each agent will have allocated a value of 1. Then the best possible NSW will be $\text{NSW}_{\text{ALG}} = (1 + M)^{k/n}$. Thus the competitive ratio will be $\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} = ((\frac{M}{1+M})^k M_n)^{1/n}$, taking $M \rightarrow \infty$ we get the desired result. \blacktriangle

Note that this sets a lower bound of $3^{1/3} \approx 1.4422$ and that the greedy has a competitive ratio of $e^{1/e} \approx 1.4447$. This result is almost tight and is connected through the function $f(x) = x^{1/x}$ that we discuss in the next section. We see that they are very near as 3 and e are near and the values are just $f(3)$ and $f(e)$.

3.2.1 The function $\exp(\ln x/x)$

We have already seen in remark 1 how this function connects with the partition of numbers. We have also seen in the greedy algorithm how we can exploit the *EF1* property to get partitions arbitrarily big that set a competitive ratio arbitrarily near the $e^{1/e}$ that we get in the continuous case.

If we examine closer the proof of lemma 6 we see that for n agents we can get a lower bound on the performance of the greedy algorithm of $(\frac{n}{m})^{m/n}$ for $m \in \mathbb{N}, m \geq n$. For n big enough this allows any approximation of $e^{1/e}$ setting $m = \lfloor n/e \rfloor$. But consider the cases $n = 2, 3, 4$. We can get $2^{1/2}$ for $n = 2, 4$ and $3^{1/3}$ for $n = 3$ as the best approximations, that match the bounds that we have for any deterministic algorithm. This hints that the greedy might be optimal in this settings, and this is in fact the case as we show in the next theorem. Although the proofs are a bit long, the flavour and technique of the proof is set in the simple case of $n = 2$.

Theorem 7. *Given $n < 5$, in the identical value setting under an online adversarial input, Algorithm 1 achieves the optimal competitive ratio of $M_n^{1/n}$, which is $n^{1/n}$ for $n < 5$.*

Proof. Without loss of generality, we can assume that the sum of the values of all items is 1 and that the biggest item has a value of B . This does not decrease any generality as NSW is invariant over multiplicative scaling. Now we separately argue for the cases when n is 2, 3, 4.

Case 1 ($n = 2$) Let the final utility of agent 1 and 2 be x and $1 - x$, respectively, such that $x \geq 1 - x$. Since the greedy algorithm induces an *EF1* allocation, their difference should satisfy $|x - (1 - x)| \leq B$. Given this constraint, the global minimum of $x(1 - x)$ is achieved when $|x - (1 - x)| = B$, and thus we obtain

$$\text{NSW}_{\text{ALG}} \geq \sqrt{x(1-x)} \geq \sqrt{\frac{1-B}{2} \frac{1+B}{2}}.$$

By the inequality of arithmetic mean and geometric mean (AM-GM), we have

$$\text{NSW}_{\text{OPT}} \leq \sqrt{y(1-y)} \leq \frac{1}{2}.$$

Now it suffices to show that $\frac{1/2}{\sqrt{\frac{1-B}{2} \frac{1+B}{2}}} = \frac{1}{\sqrt{1-B^2}} \leq \sqrt{2}$. Rearranging the inequality, this is equivalent to $B \leq \frac{\sqrt{2}}{2}$. Hence, the argument above concludes that the algorithm achieves the optimal competitive ratio when $B \leq \frac{\sqrt{2}}{2}$. Suppose now that $B \geq \frac{\sqrt{2}}{2} \geq 1/2$. In this case the optimal allocation is to allocate B to one agent and the rest of items to the other. This induces $\text{NSW}_{\text{OPT}} = \sqrt{B(1-B)}$. The competitive ratio would be

$$CR \leq \frac{\sqrt{B(1-B)}}{\sqrt{\frac{1-B}{2} \frac{1+B}{2}}} \leq \sqrt{2},$$

where the last inequality follows from the fact that it is equivalent to $(B-1)^2 \geq 0$. This completes the proof.

Case 2 ($n = 3$) The overall proof structure is similar to that of case 1. Given the constraint induced by the fact that the greedy algorithm is EF1 and the biggest item has a value of B , the minimum NSW of the greedy algorithm can be obtained by solving the following optimization problem.

$$\begin{aligned} & \underset{a, b, c}{\text{minimize}} && abc \\ & \text{subject to} && a + b + c = 1, \\ & && 0 \leq a \leq b \leq c, \\ & && c - a \leq B \end{aligned}$$

Suppose that $B \leq \frac{1}{3}$. We know that the solution of the problem should satisfy $c - a = B$. Otherwise, we could take $a - \epsilon, c + \epsilon$ for sufficiently small ϵ and obtain smaller value. Then the optimization problem can be reduced to:

$$\begin{aligned} & \underset{a, b, c}{\text{minimize}} && ab(a + B) \\ & \text{subject to} && 2a + b = 1 - B, \\ & && 0 \leq a \leq b, \\ & && b - a \leq B \end{aligned}$$

By similar argument as above, we should have $b = 1 - B - 2a$, and then it is easy to see that the minimum value will be obtained when the derivative with respect to only a becomes zero or at the boundary point of the constraint, i.e. when $b = B + a, b = a$, or if the derivative vanishes. Consider

a cubic function $f(x) = x(1 - B - 2x)(x + B)$. Note that plugging $x = \frac{1-2B}{3}$ and $x = \frac{1-B}{3}$ induces NSW_{ALG} for the cases when $b = B = a$ and $b = a$, respectively. Differentiating with respect to x , we have $f'(x) = -6x^2 + (2 - 6B)x + B - B^2$. Note that $f'(x)$ has only one positive root at $x_0 = \frac{1}{6}(1 - 3B + \sqrt{1 + 3B^2})$. It is easy to observe that x_0 lies in $[\frac{1-2B}{3}, \frac{1-B}{3}]$. Since such x_0 is the inflection point of $f(x)$ and $f'(x)$ does not have any other root in the interval $[\frac{1-2B}{3}, \frac{1-B}{3}]$, we conclude that the minimum of $f(x)$ does not belong to the interval $[\frac{1-2B}{3}, \frac{1-B}{3}]$. This implies that it suffices to show that the competitive ratio is bounded above by $3^{1/3}$ when $b = a$ or when $b = B + a$. If $b = B + a$, then we have $a = \frac{1-2B}{3}$, and $\text{NSW}_{\text{ALG}} = \frac{1-2B}{3}(\frac{1+B}{3})^2$. Otherwise if $b = a$, then we have $a = b = \frac{1-B}{3}$, and it results in $\text{NSW}_{\text{ALG}} = (\frac{1-B}{3})^2(\frac{1+2B}{3})$.

First, consider the case $b = B + a$. Due to AM-GM, the NSW of the optimal offline allocation is bounded above by $1/3$. Now it suffices to show the following.

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} \leq \frac{\frac{1}{3}}{(\frac{1-2B}{3}(\frac{1+B}{3})^2)^{\frac{1}{3}}} \leq 3^{\frac{1}{3}}.$$

By rearranging the inequality, this is equivalent to $3(1 - 2B)(1 + B)^2 \geq 1$. Define $g(x) = 3(1 - 2x)(1 + x)^2 - 1$, we can easily check that it is decreasing on $[0, \frac{1}{3}]$. Since $g(\frac{1}{3}) > 0$, this completes the proof when $B \leq \frac{1}{3}$. For case $b = a$, it is easy to see that similar argument concludes that the competitive ratio is at most $3^{1/3}$.

Now suppose that $B > \frac{1}{3}$. The optimal offline allocation will be to allocate the biggest item with B to one agent and do not allocate more items to him. Scale up the problem instances so that the summation of the valuations of all the items except the biggest one is 1. Note that this is without loss of generality again due to the scale invariance of the NSW. In this case, the biggest item has value $B' > \frac{1}{3}$. Let the value of the second biggest item be S . Following the same argument of case 1, the optimal offline allocation would result in a partition of $(B', 1/2, 1/2)$. The greedy algorithm will induce an allocation of which the NSW is lower bounded by the solution of the following optimization problem.

$$\begin{aligned} & \text{minimize} && abc \\ & && a, b, c \\ & \text{subject to} && a + b + c = 1 + B', \\ & && 0 \leq a \leq b \leq c, \\ & && c - a \leq B, \\ & && b - a \leq S \end{aligned}$$

Note that the last constraint follows from the EF1 property of the greedy algorithm. Using the similar arguments as previous, we have $c - a = B'$ at the optimum. In this case, the competitive

ratio can be obtained as

$$\begin{aligned} \frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} &\leq \left(\frac{B' \cdot \frac{1}{4}}{ab(a+B')} \right)^{\frac{1}{3}} \\ &\leq \left(\frac{1}{4ab} \right)^{\frac{1}{3}}. \end{aligned}$$

Suppose that $S \leq \frac{1}{2}$. Then, the solution of the optimization problem above can be obtained when $b - a = S$. In this case, we have $a = \frac{1-S}{3}, b = \frac{1+2S}{3}$. Hence, we conclude that

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} \leq \left(\frac{9}{4(1-S)(1+2S)} \right)^{\frac{1}{3}}.$$

To conclude that RHS is upper bounded by $(\frac{1}{3})^{\frac{1}{3}}$, by rearranging the inequality it is equivalent to show that $8S^2 - 4S - 1 \leq 0$, which is true for $S \leq \frac{1}{2}$.

On the other hand, suppose that $S > \frac{1}{2}$. In this case, the optimal allocation will be $(B, S, 1 - S)$. Using the similar arguments as we previously discussed, the NSW of the greedy algorithm is lower bounded by that of the the case where $a = \frac{1-S}{3}, b = \frac{1+2S}{3}$. Then, it suffices to show that $\frac{9(S)(1-S)}{(1-S)(1+2S)} \leq 3$. Rearranging the inequality, this is equivalent to $S^2 - 2S + 1 \geq 0$, which is always true. This completes the proof for case $n = 3$.

Case 3 ($n = 4$) Suppose that $M < \frac{1}{4}$. Then, the NSW of the greedy algorithm is lower bounded by the optimum of the following optimization problem:

$$\begin{aligned} &\underset{a, b, c, d}{\text{minimize}} && abcd \\ &\text{subject to} && a + b + c + d = 1, \\ & && 0 \leq a \leq b \leq c \leq d, \\ & && d - a \leq B \end{aligned}$$

Similarly from case 2 analysis, it is straightforward to check that $d = a + B$ and $c = d$ or $b = a$, and that the minimum point of the resulting optimization problem does not have its optimum in the interior point. This again implies that the minimum value of NSW will be achieved by $(x, x + M, x + M, x + M)$ for $x = (1 - 3M)/4$, $(x, x, x + M, x + M)$ for $x = (1 - 2M)/4$ or $(x, x, x, x + M)$ for $x = (1 - M)/4$. Note that the optimal offline allocation has NSW of $1/4$ by the allocation $(1/4, 1/4, 1/4, 1/4)$. Repeating the similar process as in case 2, when $M \leq 1/4$, we can easily conclude that the competitive ratio is at most $\sqrt{2}$. Suppose that $M > 1/4$. Then this large-valued item has to be allocated alone in any optimal offline allocation. Denote the value of the second biggest item by N . Without loss of generality, assume that the total sum of values over the items is $1 + M$. Obviously the optimal offline allocation is $(M, 1/3, 1/3, 1/3)$. The worst possible allocation that may come from the greedy algorithm is $(x + M, x + N, x + N, x)$ or $(x + M, x + N, x, x)$

for the x that makes the sum of values equal $1 + M$. Note that the competitive ratio is an increasing function over M . Thus it suffices to show that both $(\frac{1}{3^3(\frac{1-2N}{4}+N)^2\frac{1-2N}{4}})^{1/4}$ and $(\frac{1}{3^3(\frac{1-N}{4}+N)(\frac{1-N}{4})^2})^{1/4}$ are upper bounded by $\sqrt{2}$. Followed by some elementary level calculations, we can easily argue that these quantities are upper bounded by $\sqrt{2}$ for $N \leq 1/3$. Suppose that $N > 1/3$. Without loss of generality, we can scale all the values so that the sum of item values are $1 + M + N$ where there are two big items of $M' > M$ and $N' > N > 1/3$. In this case, the optimal offline allocation again will be $(M', N', 1/2, 1/2)$. Denote by S the value of third biggest item. By repeating similar arguments as previous, we can easily argue that the NSW of the greedy algorithm is lower bounded by the allocation $(M' + x, N' + x, S + x, x)$ for $x = (1 - S)/4$. Again as the competitive ratio over this allocation and the optimal offline allocation of $(M', N', 1/2, 1/2)$ is an increasing function over both M' and N' , the competitive ratio of the greedy algorithm is eventually upper bounded by the case when M' and N' goes to ∞ , which results in

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} \leq \left(\frac{16}{2^2(1-S)(1+3S)} \right)^{1/4},$$

and we want to show that this quantity is bounded above by $\sqrt{2}$. By rearranging the inequality, this is equivalent to $-3S^2 + 2S \geq 0$, and this is true for $S \in [0, 2/3]$. Finally, suppose that $S > 2/3$, then $S \geq 1/2$. Again by scaling the problem instance, suppose that the optimal offline allocation is achieved at $(M, N, S, 1)$. Similar argument leads us to conclude that the worst possible greedy allocation is $(M + 1/4, N + 1/4, S + 1/4, 1/4)$, which gives us the competitive ratio of $\sqrt{2}$, and this completes the proof. We finally note that the bound is tight only if there exists two or three large-valued items, which is consistent with our worst-case instance in the lower bound. \blacktriangle

The previous theorem proves that in the family of deterministic algorithms we can have $EF1$ and optimal NSW maximization under a low number of agents. This is not obvious for $n > 4$ and can not be deduced from our work here, as we would need to improve our lower bound (that might be optimal). For completion we note that the previous result can not be extended for $n = 5$. Indeed, consider an instance such that 10 items with value 1 arrives, and then 3 items with value x arrive. By making x sufficiently large, the competitive ratio of the algorithm becomes close to $(\frac{5}{2})^{2/5}$, which is strictly larger than $6^{1/5}$.

3.3 Small items case

In the previous proof we mainly used one idea: if we know the largest valuation we can bound from below the NSW of the greedy algorithm. This allows us ensure good competitive ratios if this largest item is small enough.

Note that the worst cases of the online greedy and the lower bounds on the competitive ratio are met in the presence of big items that are valued infinitely more than the rest of items. This is indeed not the expected case in a real world situation. This pushes us to consider the greedy algorithm under the assumption that no item is valued too much. Namely, the valuation V of any

item is smaller than fT where T is the sum of all valuations and $f \in [0, 1]$ is a parameter that constraints how large can the largest item be. In this setting we can derive an upper bound on the competitive ratio.

Theorem 8. *In the equal valuations setting under adversarial input, if for all every item I we have $v(I) \leq fT$ where T is the sum of valuations and $f \in (0, \frac{1}{n})$ we have that the competitive ratio of the greedy algorithm is at most $(\sqrt{1 - fn} + nf/2)^{-1}$.*

Proof. Without lose of generality take $T = 1$. As in the previous proof we will use the fact that under equal valuations the total allocated value by the greedy to two different agents differs at most the value of the greatest item. Then in the final allocations of items if v_i is the value for the i -th agent for all i, j we have $|v_i - v_j| \leq f$.

We will suppose that the optimal allocation gets the maximum possible value $\text{NSW}_{\text{OPT}} = 1/n$ when $v_i = 1/n$ for all i and we will see which is the minimum possible value of $\text{NSW}_{\text{ALG}} = (v_i)^{1/n}$ under the condition $\sum v_i = 1$ and $|v_i - v_j| \leq f$. Note that if $f \geq \frac{1}{n-1}$ the allocation $v_1 = 0$ and $v_i = \frac{1}{n-1}$ for $i \neq 1$ gets a NSW of 0.

Let x be the minimum received utility through all agents. Now in the worst case we have that $v_i \in \{x, x + f\}$ for all but at most one agent. If for $i \neq j$ we had $v_i, v_j \in (x, x + f)$ we could take $v'_i = v_i + \epsilon, v'_j = v_j - \epsilon$ for an small enough ϵ obtaining an smaller NSW. Then we can assume that we have l small agents with utility x , $n - l - 1$ big agents with utility $x + f$ and one agent with utility $y = \lambda x + (1 - \lambda)(x + f)$ for some $\lambda \in [0, 1]$. If we loose our condition to have $l \in [1, n - 1]$ instead of $l \in [n - 1]$ we can assert that λ is either 0 or 1 and then we can suppose that we have l small and $n - l$ agents. We see this because setting $l' = l + \lambda$ gets an smaller NSW: $x^{l'}(x + f)^{n-l'-1}y \geq x^{l'+1}(x + f)^{n-(l'+\lambda)-1} \iff \lambda x + (1 - \lambda)(x + f) \geq x^\lambda(x + f)^{1-\lambda}$. The former is true because it is true for $f = 0$ and taking derivatives on f we have that $(1 - \lambda) \geq (1 - \lambda)x^\lambda(x + f)^{-\lambda} \iff (x + f)^\lambda \geq x^\lambda$ that is true since $\lambda \geq 0$.

We can now relate x and l using that the total utility is 1. For simplicity we say that there are l big agents and $n - l$ small agents. Then $x(n - l) + (x + f)l = 1 \implies x = \frac{1 - fl}{n}$. Then $\text{NSW}_{\text{ALG}} \geq \left(\left(\frac{1 - fl}{n} \right)^{n-l} \left(\frac{1 + f(n-l)}{n} \right)^l \right)^{\frac{1}{n}} = \frac{(1 - fl)}{n} \left(1 + \frac{fn}{1 - fl} \right)^{l/n}$. Now we will bound the former using the Taylor expansion of $(1 + x)^\lambda$, using the following claim:

Claim 5. *For $x \geq 0$ and $0 \leq \lambda \leq 1$ we have $(1 + x)^\lambda \geq 1 + x\lambda + x^2\lambda(\lambda - 1)/2$*

Proof of the Claim. Call $L(x)$ to the LHS of the inequality and $R(x)$ to the RHS. We have that $L(0) = R(0)$ thus it is enough to see $L'(x) \geq R'(x)$ for $x \geq 0$. $L'(x) = \lambda(1 + x)^{\lambda-1}$ and $R'(x) = \lambda + x\lambda(\lambda - 1)$. Since $L'(0) = R'(0)$ it is enough to see that $L''(x) \geq R''(0)$ for $x \geq 0$. We have $L''(x) = \lambda(\lambda - 1)(1 + x)^{\lambda-2}$ and $R''(x) = \lambda(\lambda - 1)$. Using that $\lambda(\lambda - 1) < 0$ we have $L''(x) \geq R''(x) \iff (1 + x)^{\lambda-2} \leq 1$ that is true. \blacktriangle


Using the claim we have $\text{NSW}_{\text{ALG}} \geq \frac{1 - fl}{n} \left(1 + \frac{fn}{1 - fl} \frac{l}{n} + \frac{(fn)^2}{(1 - fl)^2} \frac{l/n(l/n - 1)}{2} \right) = \frac{1}{n} \left(1 + \frac{f^2 l(l - n)}{2(1 - fl)} \right)$. We are left to minimize $g(l) = \frac{l(l - n)}{1 - fl}$. Taking the derivative we find the minimum at $l = \frac{1 - \sqrt{1 - fn}}{f}$. Substituting we get that $\text{NSW}_{\text{ALG}} \geq \frac{1}{n}(\sqrt{1 - fn} + nf/2)$ and then the $CR \leq (\sqrt{1 - fn} + nf/2)^{-1}$



Remark 2. *The worst case used in the previous proof can be reached. Just begin with a lot of small items (so in the online setting it is possible to get the $1/n$ NSW) and then receive the items valued fT . Thus the given value would be the exact competitive ratio if it was not because of the Taylor approximation.*

This last theorem bound seems a bit difficult to contextualize. We get some insight on the next corollary, observing that making nf constant simplifies the expression.

Corollary 1. *When $f \leq \frac{1}{kn}$ for a $k \geq 1$ we can assert a competitive ratio of at most $(\sqrt{1 - \frac{1}{k}} + \frac{1}{2k})^{-1}$.*

Proof. Substitute $f = \frac{1}{kn}$ in the previous theorem bound. 

For example, the previous corollary with $k = 2$ (impose that no item has more than $1/(2n)$ of the total utility) asserts a competitive ratio of at most $\frac{4}{1+2\sqrt{2}} \approx 1.044$ much lower than the ≈ 1.442 general bound.

3.4 The intrinsic difficulty

In the lower bound proof of Theorem 6 we used that any deterministic algorithm has to allocate the first n items to different agents if it does not lack from a infinite competitive ratio, creating a distribution of the allocated values that is too “flat”.

We want to emphasize that this problem about the “shape” of the distribution of the allocated values is absolutely central in the design of any algorithm for this problem. Let see the importance of this distribution.

Consider an algorithm that has received a set of items S and allocated a total utility of a_i to agent i and suppose that $a_i \geq a_{i+1}$ and $\sum a_i = 1$. Now we consider the arrival of $b \in [0, n - 1]$ big items with value X that we will take arbitrarily large. As in the proofs of the previous lower bounds this enormous objects make insignificant the previously allocated value of the agent that they are allocated to. Suppose that we receive $b = n - l$ of this big objects. Call $P(S, l)$ to the optimum allocation of the items in S to l different agents in order to maximize the multiplication of the allocated values. Then we have that $\text{NSW}_{\text{OPT}} = (P(S, l)X^{n-l})^{1/n}$ and $\text{NSW}_{\text{ALG}} = \prod_{i=1}^l a_i \prod_{i=l+1}^n (X + a_i)^{1/n}$. Taking $X \rightarrow \infty$ we have that $CR(l) = \frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} = \left(\frac{P(S, l)}{\prod_{i=1}^l a_i} \right)^{1/n}$. If we suppose that $P(S, l)$ can take the optimal value of $(\frac{1}{l})^l$ and we want to have a competitive ratio of at most c we need that for all $l \in [n]$:

$$CR(l) = \left(\frac{(\frac{1}{l})^l}{\prod_{i=1}^l a_i} \right)^{1/n} \leq c \iff \prod_{i=1}^l a_i \geq \frac{(\frac{1}{l})^l}{c^n} \iff \sum_{i=1}^l \log a_i \geq -l \log l - n \log c$$

. Unluckily this system of inequalities is not linear so we can not use the same techniques we saw in Section 1.3.1 to obtain an optimal distribution of a_i and an optimal c . We discuss a couple of simple examples for understanding the hardness of the problem before solving the problem:

We already saw in the lower bound results section the problem with a flat distribution. Take $a_i = \frac{1}{n}$ it is optimum for $l = 0$ and good for small l . But if we take $l = \lfloor \frac{n}{e} \rfloor$ we get $c \gtrsim \left(\frac{(\frac{\epsilon}{n})^{n/e}}{(1/n)^{n/e}} \right)^{1/n} = e^{1/e}$ when $n \rightarrow \infty$, that is not optimal as we shall see.

In the other hand we can take a non-flat distribution to make sure that we have utility allocated in the first a_i so we do not run in the previous problem. But in this case we might end with an allocation that has a poor geometric mean of all the a_i together. Take for example a geometrically decreasing distribution $a_i = e^{-ti} \frac{e^t - 1}{e^t - e^{-tn}}$ for $t > 0$. In this case taking $l = 0$ we get $c \geq \frac{1/n}{(\prod_{i=1}^n e^{-ti} \frac{e^t - 1}{e^t - e^{-tn}})^{1/n}} = \frac{e^t - e^{-tn}}{e^t - 1} \frac{e^{(n+1)t/2}}{n}$ that gets arbitrarily big when $n \rightarrow \infty$.

3.4.1 Optimal distribution

Before jumping to the problem let's fix some notation. We are interested in the next optimization problem:

$$\begin{aligned} & \text{minimize} && c \\ & \text{subject to} && A_i : -\sum_{j=1}^i \log a_j - i \log i \leq n \log c \quad \forall i \in [n], \\ & && \sum_{i=1}^n a_i = 1, \\ & && a_i \geq a_{i+1} \geq 0 \quad \forall i \in [n-1] \end{aligned}$$

We can find an approximated solution with an iterative method, that will hint us to an analytical solution of the problem. For doing so we initialize the a_i at random and we run iterations where we calculate the value of each LHS of the A_i conditions. If j is the $\text{argmax}_i LHS(A_i)$ we make an update $a'_j = a_j + \epsilon - \epsilon/n$ and $a'_i = a_j - \epsilon/n$ for a small ϵ this will decrease the $LHS(A_i)$ as we are allocating more utility to the agent a_i . Making this updates iteratively we expect to converge to the optimal solution (as we will see that happens). The code of this iterative solver can be found on Appendix A. Some results of the iterative method are in Figure 3.1. We can observe some things about the numerical solution: the first a_i are equal and approximately $2/n$ and in the point that the A_i conditions become tight they stop being constant and take a value $\approx \frac{1}{ie}$. We also see that $c \approx 1.2$. We will see how this holds in the following lemmas.

In the following will talk about A_i , a_i and c as they are the values in the optimal solution.

Lemma 8. *If A_i is tight so is A_{i+1} .*

Proof. For the sake of contradiction suppose $A_i : -\sum_{j=1}^i \log a_j - i \log i = n \log c$ and $A_{i+1} : -\sum_{j=1}^{i+1} \log a_j - (i+1) \log(i+1) < n \log c$. Subtracting A_i from A_{i+1} we get $-\log a_{i+1} < (i+1)$

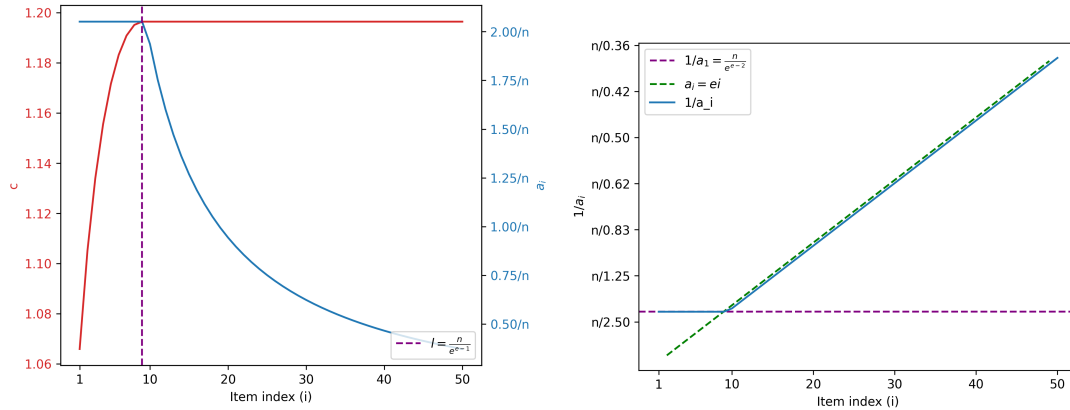


Figure 3.1: The figures are the numerical results of the iterative optimization problem with $n = 50$. We get $c = 1.196$ and $a_1 \approx 2.051/n$ the values also found in the proofs of the section.

In the left one in red it is the blue c needed to satisfy inequality A_i for each i and in blue the utility allocated in a_i . It can be seen that c stabilises exactly when a_i starts to drop. We can also see that for low i only a small c is needed, this can be expected as a_1 will always be at least $1/n$ by the pigeonhole principle and $n^{1/n} \rightarrow 1$ when $n \rightarrow \infty$.

In the left we can look at $1/a_i$ which clues about the distribution of a_i . We see that when the a_i start to vary they are almost equal to ei as we shall formally prove.

1) $\log(i+1) - i \log i \iff a_{i+1} > \left(\frac{i}{i+1}\right)^i \frac{1}{i+1}$. Note that then we can just make $a'_{i+1} = \left(\frac{i}{i+1}\right)^i \frac{1}{i+1}$ that makes A_{i+1} tight. We can add this extra utility to a_{i+2} and since $a_{i+2} \leq a_{i+1}$ the A_j with $j > i+1$ will also be satisfied. Since all conditions have stayed equal or improved we can assume that the statement is true in the optimal solution. \blacktriangle

This implies that there exists an $l \in [n-1]$ such that for all $i > l$ we have that A_i is tight and that for all $i \leq l$ A_i is slack. Note that this holds since at least one of the conditions is tight, since if this does not hold c could be reduced. We see that we then calculate the value of a_i for $i > l$.

Lemma 9. *If A_i and A_{i+1} are tight $a_{i+1} \approx \frac{1}{e(i+1)}$.*

Proof. We have $A_i : -\sum_{j=1}^i \log a_j - i \log i = n \log c$ and $A_{i+1} : -\sum_{j=1}^{i+1} \log a_j - (i+1) \log(i+1) = n \log c$. Subtracting these expressions we get: $\log a_{i+1} = i \log i - (i+1) \log(i+1) \iff a_{i+1} = \left(\frac{i}{i+1}\right)^i \frac{1}{i+1} \approx e^{-1} \frac{1}{i+1}$. We will see that when $n \rightarrow \infty$ also holds $l \rightarrow \infty$ so the approximation will be arbitrarily tight for all $i > l$. \blacktriangle

Then we have derived the value for all the i where the A_i are tight. We will now see that for the rest of i the value of a_i is constant.

Lemma 10. *For all $i, j \in [l]$ it holds $a_i = a_j$.*

Proof. Let j be $\max_{i \in [l-1]} a_i \neq a_{i+1}$. Note that since A_j is slack and $a_j \neq a_{j+1}$ we can pick a small enough ϵ and make $a'_j = a_j - \epsilon$, $a'_{j+1} = a_{j+1} + \epsilon$ while it still holds $a'_j \neq a'_{j+1}$ and that A'_j is still slack. All the A_i with $i < j$ are still slack since we did not change anything. Note now that

$-\log a_j - \log a_{j+1} \geq -\log a'_j - \log a'_{j+1}$ since $-\log a_j - \log a_{j+1} \geq -\log(a_j - \epsilon) - \log(a_{j+1} + \epsilon) \iff \log(a_j a_{j+1}) \leq \log(a_j a_{j+1} + \epsilon(a_j - a_{j+1}) - \epsilon^2) \iff \epsilon \leq a_j - a_{j+1}$ and since $a_j > a_{j+1}$ we can take ϵ to full fill it. The previous inequality implies that for all $i > j$ the *LHS* of A_i has decreased. In particular those inequalities that were tight now they are slack which contradicts the optimality of the distribution. \blacktriangle

We are only left to determine the values of a_i for $i \leq l$ and l . For doing so we will find the l that optimizes c , since we shall see in next lemma that the two unknowns are strongly related.

Lemma 11. *The first $l \approx \frac{n}{e^{e-1}} \approx 0.17n$ a_i are equal with value $\approx e^{e-2}/n \approx 2.05/n$.*

Proof. Begin with the condition that we have still not used $\sum a_i = 1$. we have $\sum a_i = \sum_{i \leq l} a_i + \sum_{i > l} a_i = a_1 l + \frac{1}{e} \sum_{i > l} \frac{1}{i} = a_1 l + \frac{H_n - H_l}{e}$ where H_n is the n -th harmonic number. Using the approximation $H_n \approx \log n$ (note that the Euler-Mascheroni constant wipes out as we are subtracting harmonic numbers) we have: $a_1 l + \frac{\log(n/l)}{e} = 1 \implies a_1 = \frac{1 - \log(n/l)/e}{l}$. Note now that A_l is approximately tight (we could do the calculations with A_{l+1} or A_n but we take A_l for simplicity). Then we are interested in minimizing

$$LHS(A_l) = la_1 - l \log l = -l \log \left(\frac{1 - \log(n/l)/e}{l} \right) - l \log l = -l \log(1 - \log(n/l)/e)$$

Taking the derivative we need

$$-LHS(A_l)' = \log(1 - \log(n/l)/e) + \frac{1}{e - \log(n/l)} = 0$$

Taking $n/l = e^t$ it simplifies to $\log(1 - t/e) = \frac{1}{t-e}$ and it is easy to see that $t = e - 1$ gives a solution (one can check the uniqueness seeing that it is the only minima for $0 < t < e$). Then we have that $n/l = e^{e-1} \implies l = \frac{n}{e^{e-1}}$ and $a_1 = \frac{1 - (e-1)/e}{n/e^{e-1}} = \frac{e^{e-2}}{n}$. \blacktriangle

Corollary 2. *The a_i with $i > l \approx 0.17n$ have value $a_i = \frac{1}{ei}$*

Theorem 9. *In the optimal distribution we have that $c \approx e^{1/e^{e-1}} \approx 1.196$.*

Proof. It follows from the previous lemmas. We only have to plug the value of l in A_l as seen in Lemma 11 and we get:

$$LHS(A_l) = -\frac{n}{e^{e-1}} \log(1/e) = \frac{n}{e^{e-1}} = n \log(c) = RHS(A_l) \implies c = e^{1/e^{e-1}}$$

\blacktriangle

Note that all this sets a fundamental bound: even if an online algorithm has been running for a lot of time and has been able to allocate the utilities in the best possible distribution, the arrival of several big items can make the competitive ratio jump to at least 1.196. In the next section we will see a bound of the cost of allocating the initial items with randomized algorithms that will set a bigger ratio, as it usually not possible to achieve this optimal distribution.

3.5 Randomized algorithms

3.5.1 Lower bound

The strong lower bound of $3^{1/3}$ only applied to deterministic algorithms. In this section we will see how the same construction can be used to derive a lower bound that also applies to randomized algorithms. We will use the idea explained in the introduction about seeing random algorithms as a distribution of deterministic algorithms. This helps us to rephrase the study of the random algorithms as an study on the distribution in the deterministic algorithms space, that allows us to find system of linear inequalities to derive lower bounds on the performance.

In the following theorem we use the same input that in the deterministic lower bound: we receive n items valued equally followed by some number of big items (between none and $n - 1$). We extract from a random algorithm the probability distribution of the number of agents that received some item of the first n items. Note that in the deterministic case we knew that every agent should receive an item.

Theorem 10. *In the identical value setting with adversarial input, any (possibly randomized) algorithm has a competitive ratio of at least 1.3692.*

Proof. Recall the techniques used in Lemma 1. Consider any algorithm ALG and call $\{p_i\}_{[n]}$ the probability that it allocates items to i different agents if it receives n items valued at 1 as the first items. Consider then the competitive ratio that it will get in the set of inputs $\{I_i\}_{0\dots n-1}$ where the input I_i consists on n items valued at 1 and i arbitrarily large items. Call C_i to the competitive ratio on input I_i and C_{ALG} to the maximum C_i . Then the competitive ratio of any algorithm is bounded by the minimum C_{ALG} on all the possible distributions $\{p_i\}_{[n]}$. In short we are interested in $\min_{\{p_j\}_{[n]}} \max_i I_i(\{p_j\}_{[n]})$.

This optimization problem can be rewritten as an LP problem using the following claim.

Claim 6. *In the optimum of the previous optimization problem we have that $C_{\text{ALG}} = \frac{1}{p_n}$ that is the competitive ratio on the input I_0 .*

Proof. Firstly, rephrase the problem to maximize the inverse of the competitive ratios:

$\max_{\{p_j\}_{[n]}} \min_i 1/I_i(\{p_j\}_{[n]})$. Note then that for I_0 we have that $1/C_0 = p_n$ as we if not some agent will not receive any item the NSW will be 0. For all other inputs I_i for $i > 0$ inputs allocating the first items to $n - 1$ different agents gives a NSW of $2^{1/n}$ times the NSW of allocating them to the n agents. Suppose now that $\min 1/C_i < 1/C_0 = p_n$. Then we could set $p'_n = p_n - \epsilon$ and $p'_{n-1} = p'_{n-1} + \epsilon$ for an small $\epsilon > 0$ that satisfies $p'_n > 1/C_i$. This operation will improve $\min 1/C_i$. Then we found a bigger minimum contradicting the fact that the previous algorithm was optimal. Then we know than $\min 1/C_i = 1/C_0$. ▲

Now we can create an LP problem where we are interested in maximizing $1/C_0 = 1/p_n$ with the restriction that $1/C_0 \leq 1/C_i$ for all i .


An example with $n = 5$ can be found on figures 3.2 and 3.3.

Agents w/ items \ Big items	1	2	3	4	5
0	0	0	0	0	1
1	0	0	0	$2X$	X
2	0	0	$4X^2$	$2X^2$	X^2
3	0	$6X^3$	$4X^3$	$2X^3$	X^3
4	$5X^4$	$4X^4$	$3X^4$	$2X^4$	X^4

Figure 3.2: For $n = 5$ if the value of the big item is X it is shown the maximum possible product of the utilities for each number of agents receiving some of the first $n = 5$ 1-valued items.

$$\begin{aligned}
& \text{minimize} && C_0 = 1/p_5 \\
& \text{subject to} && C_1 = \frac{2^{1/5}}{2^{1/5}}p_4 + \frac{1}{2^{1/5}}p_5 \geq p_5 = C_0, \\
& && C_2 = \frac{4^{1/5}}{4^{1/5}}p_3 + \frac{2^{1/5}}{4^{1/5}}p_4 + \frac{1}{4^{1/5}}p_5 \geq p_5 = C_0, \\
& && C_3 = \frac{6^{1/5}}{6^{1/5}}p_2 + \frac{4^{1/5}}{6^{1/5}}p_3 + \frac{2^{1/5}}{6^{1/5}}p_4 + \frac{1^{1/5}}{6^{1/5}}p_5 \geq p_5 = C_0, \\
& && C_4 = \frac{5^{1/5}}{5^{1/5}}p_1 + \frac{4^{1/5}}{5^{1/5}}p_2 + \frac{3^{1/5}}{5^{1/5}}p_3 + \frac{2^{1/5}}{5^{1/5}}p_4 + \frac{1}{5^{1/5}}p_5 \geq p_5 = C_0, \\
& && \sum_{i=1}^n p_i = 1, \\
& && p_i \geq 0 \quad \forall i \in [n]
\end{aligned}$$

Figure 3.3: System to optimize for $n = 5$ obtained with the NSW computed with Table 3.2.

This LP can be created and solved for any n with the code provided in Appendix B. Solving it for $n = 1500$ gives a lower bound of ≈ 1.3692 on the competitive ratio of any randomized algorithm. 

3.5.2 A proposal of a randomized algorithm

In this section we use the knowledge found in this chapter to propose a randomized algorithm. We support it's convenience through simulations on the achieved competitive ratio versus some selected inputs. With this we want to provide empirical evidence that there exists a randomized algorithm that breaks the deterministic lower bound.

We propose a family of algorithms called 2-bucket greedy that their general outline is Algorithm 2. We still have to define what is the distribution $\{p_i\}$ for each n and the function f used to decide to which bucket to allocate a given item.

The idea of this family of algorithms is to maintain two type of agents. The *big* and the *small*

agents. We allocate the regular items to the “big” agents and we allocate to the “small” agents only the items with big valuations. With this we try to ensure that our allocation is never too plain so we can receive big items without getting a bad competitive ratio as we seen in the regular greedy. But we also use the acceptable good performance of the greedy in two ways. The first is for allocating the items within the groups. The second is that with a certain probability we will run the regular greedy. The other probabilities give a certain weight to the different sizes that the big and small items sets can have, so we can hedge for cases with a lot of big items present or with a only a few.

Algorithm 2: 2-bucket greedy

```

1 Pick  $D \in [n]$  following a given distribution  $\{p_i\}_{i \in [n]}$ ;
2 if  $D = n$  then
3   | Run the regular greedy algorithm 1;
4 end
5 else
6   | Maintain allocations  $B_i \in [D]$  and  $S_i \in [n] \setminus [D]$  such that  $v(B_i) > v(S_j) \forall i, j$ .
7   for  $t = 1, 2, \dots, T$  do
8     | Item  $g_t$  arrives ;
9     |  $b = \sum_{i \in [D]} v(B_i)$ 
10    |  $s = \sum_{i \in [n] \setminus [D]} v(S_i)$ 
11    | if  $v(g_t) \geq f(b, s, n)$  then
12      | Allocate item  $g_t$  greedily in  $S$ ;
13    | end
14    | else
15      | Allocate item  $g_t$  greedily in  $B$ ;
16    | end
17    | Update  $S$  and  $B$  in order of maintaining the invariant;
18  | end
19 end

```

We have hence to define $\{p_i\}$ and f . For f we will show results with several different functions, discussing more in depth the one that gives better results. The main idea will be to use only the sum of the utilities of one of the groups, weighted by some factor, to compare versus the new arriving item and deciding by that.

For the distribution we will follow an “inverse” strategy: we will pick the $\{p_i\}$ that minimizes our competitive ratio. This is possible to do since in our experiments we will have a finite amount of input cases. But it is acceptable, as we aim to show that it is reasonable to think that a good randomized algorithm exists. This maximization can be done solving a linear program, similar to what we did in the previous section for finding a lower bound on the competitive ratio.

The other important decision will be what inputs should we use for computing the competitive ratio. In order to have meaningful results we need to have strong inputs. The main problem with

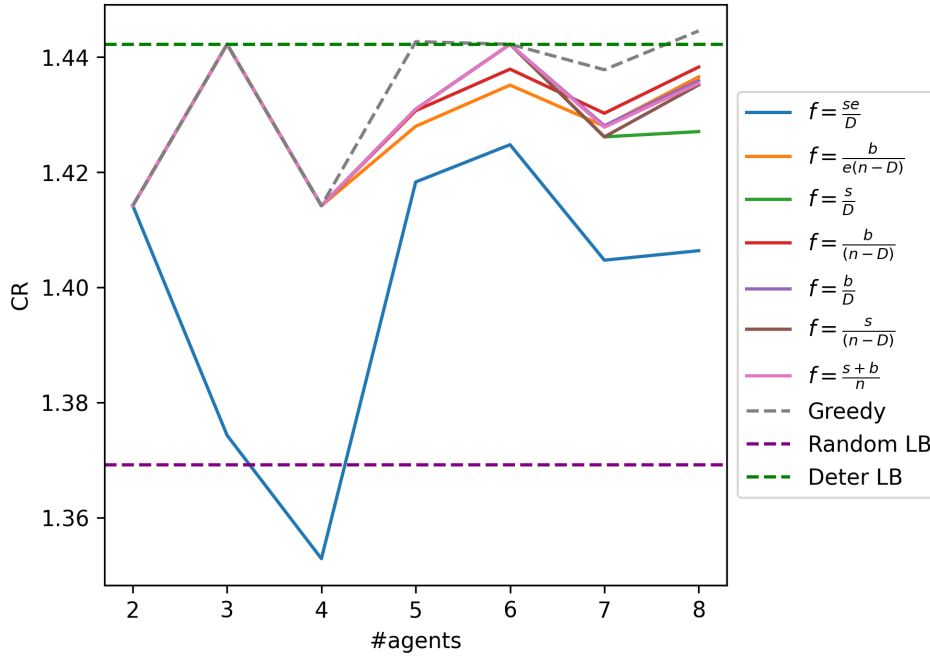


Figure 3.4: Competitive ratio for several agents with several decision functions and options `--perms=500` `--approximation`, the approximation function approximates the optimal NSW of some difficult cases giving an upper bound of it, so the CR obtained are also upper bounds. b stands for the sum of the “big” items and s for the sum of the “small” items. We see that the function $f = se/D$ gets a significantly better performance than the rest. Most surprisingly with only 3 agents it is the only algorithm that outperforms the greedy, that in that case we shown to be optimal for a deterministic algorithm.

this is that given an arbitrary input calculating it’s optimal NSW is a NP-hard problem as we already discussed. Then we can not just generate a big amount of random cases and expect to have strong cases, as we would need too much time for calculating the optimal value of each of those cases. We will then use hand designed cases. We will focus on the inputs that we shown to be the bad inputs in the lower bounds proofs, alternating small and very big items. We will also use some other inputs that have closed-form optimum allocations. We will input permutations of all this kind of inputs in order to get more “cheap” inputs. All the inputs can be checked in the implementation on Appendix C.

In figure 3.4 we see how picking $f = \frac{se}{D}$ looks like the right decision. Moreover one can also check that the balance factor e is relevant in the definition. In this algorithm we allocate an item valued v in the “small” bunch S if $v(n - |S|) \geq e \sum_{s \in S} v(s)$. This is, if in the case that each agent of the other bunch had an item valued v the value allocated in that bunch would be greater to the total allocated value in S times a normalization constant.

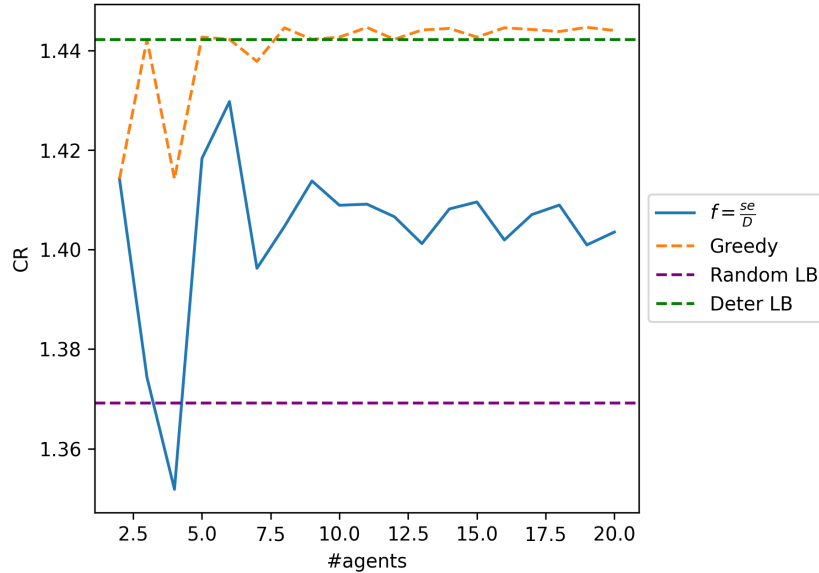


Figure 3.5: Results of the experiment with options `--perms=100` `--approximation` with the best function found. We can see how the competitive ratio stabilizes around 1.41 having a significant better performance than the best possible deterministic algorithm.

We could expect the algorithm to behave just as the greedy most of the times (this is $p_n \approx 1$) as it works well in the general case and we only want to hedge the worst cases with the other cases of the distribution. We can see the distribution in figure 3.6 and the simulation with more agents in figure 3.5. Looking at this 2 figures it is heavily suggested that our algorithm is capable of outperforming any deterministic algorithm by choosing the right distribution of D and hedging with that the bad cases coming from an arrival of unexpected big items.

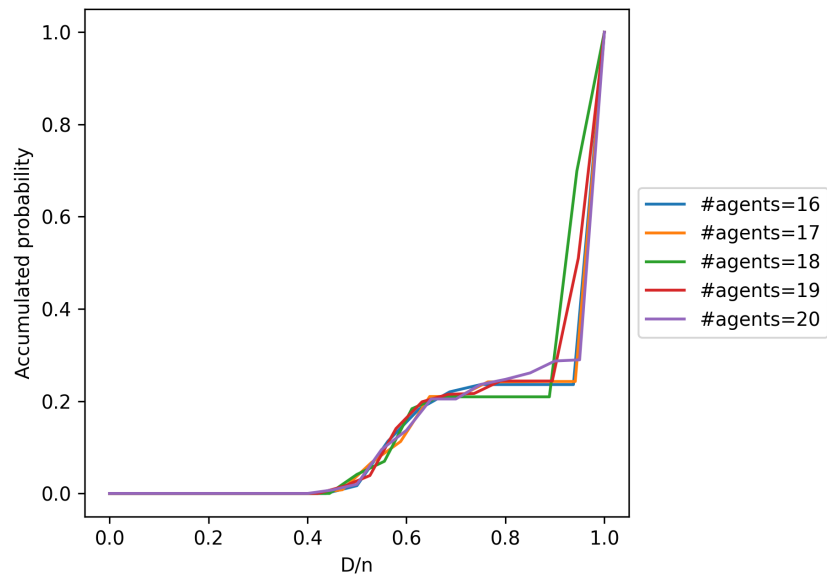


Figure 3.6: Distribution of the probabilities on D for different number of agents, $f = se/D$ and options `--perms=100` `--approximation`. We show the accumulated probability of picking a D smaller than x , normalized by dividing by the number of agents $= n$. We can see how the D smaller than $0.5n$ have probability near to 0 of being picked. We further see that the $D = n$ that corresponds to the regular greedy algorithm gets a probability between 0.5 and 0.8, but not more than that, so the algorithm is effectively using the other values of D for improving the performance as we have seen in the previous figures. Finally we observe a curious effect where the $D \in [0.65n, 1n)$ are picked with a very low probability, against the intuition that since they are similar to the greedy algorithm they would be picked more often.

4. Affine utility setting

We here proceed by generalizing the identical setting of Section 3 through *affine valuations* where the valuation of an item to each agent is an affine transformation of the base utility function.

Definition 6 (Affine value). *Given a base utility function $u(\cdot)$ and constants $a_i > 0, b_i \geq 0$, we define an **affine** value setting to be the case in which agent i 's valuation function v_i of receiving an item t satisfies $v_i(\{t\}) = a_i u(\{t\}) + b_i$.¹*

We highlight that the affine utility function is motivated by the existence of a common preference relation over the items. Consider a scenario in which there exists a preference relation (a total ordering) over the set of items and all the agents have the same preferences over the items. For example, while natural resources like gold, silver, and copper may have varying monetary values in different currencies, the total order of the nations using these currencies is consistent. Namely, even though the exact valuations for the items can be different for each agent, their valuations are equal up to the corresponding currency rate, which often is a linear transformation.

It is widely known that if there exists a utility function² that represents this preference relation, then it is unique up to the affine transformation Mas-Colell *et al.* [1995]. This implies that the affine utility setting is effectively the scenario in which all the agents share a preference relation over the items.

In this setting, we have a pair of parameters a_i, b_i for each $i \in [n]$ and the algorithm is aware of these parameters in advance. This restriction is indeed a mild assumption since, even when $(a_i, b_i)_{i \in [n]}$ are not revealed in advance, each agent's exact valuation with respect to the arriving item is revealed before the algorithm allocates the item in our problem setting. We can thus recover all the parameters after some initial rounds by solving a system of linear equations. Moreover, we note that for $a_i = 1$ and $b_i = 0$ we exactly recover the identical valuation setting. This implies that the obtained lower bounds of Section 3 carry over in this setting as well.

Before presenting our main results, we show that it suffices to consider only the case in which $a_i = 1$ for all $i \in [n]$.

Theorem 11. *In the affine value setting, any problem instance with parameters $(a_i, b_i)_{i \in [n]}$, can be reduced to the problem instance with $(1, b'_i)_{i \in [n]}$.*

Proof. We use the fact that the NSW problem is invariant up to scaling of an agent's valuation for all items. Given a set of items \mathcal{M} and n agents, we scale the i -th agent valuation function v_i (without loss of generality) by some constant factor $s > 0$. Any allocation, $X = \{X_1, \dots, X_n\}$, that previously obtained valued $v_i(X_i)$ for the given agent will now yield $s \cdot v_i(X_i)$. The NSW objective

¹It is often denoted by cardinal utility function in the literature.

²Precisely, this has to be Von Neumann–Morgenstern utility function to guarantee the uniqueness up to the affine transformation.

will correspondingly increase by a factor of $s^{1/n}$ and the competitive ratio of this allocation will remain unchanged.

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} \Rightarrow \frac{s^{1/n}\text{NSW}_{\text{OPT}}}{s^{1/n}\text{NSW}_{\text{ALG}}} = \frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}}$$

We can thus multiply every agent's valuation by $1/a_i$ and the theorem is proven. \blacktriangle

We now present a lower bound result for any deterministic algorithm in an $n = 2$ agent system which further implies that randomization is required to guarantee a non-trivial competitive ratio in expectation.

Theorem 12. *In the affine value setting, for any deterministic algorithm and constant $M \geq 1$, there exists a problem instance with $n = 2$ such that the algorithm has a competitive ratio larger than M .*

Proof. Consider a deterministic algorithm for two agents with $(a_1, b_1) = (1, 1)$ and $(a_2, b_2) = (1, 0)$. We define two problem instances, \mathcal{I}_1 and \mathcal{I}_2 , both on two items denoted t_1 and t_2 . In \mathcal{I}_1 , the items arrive with utilities $u(t_1) = \epsilon > 0$ and $u(t_2) = 0$ while in \mathcal{I}_2 the utilities are ϵ and $c > 0$ respectively. Since the algorithm is deterministic, its induced allocation must be indifferent for the two problem instances \mathcal{I}_1 and \mathcal{I}_2 at the arrival of the first item. Suppose the algorithm allocates item t_1 to the first agent. If the second item arrives with respect to instance \mathcal{I}_1 , then the NSW of the algorithm must be 0 and the optimal allocation is revealed to be allocating t_1 to agent 2 and t_2 to agent 1. Thus, the algorithm has competitive ratio of $\sqrt{\epsilon \cdot 1}/0 = \infty$.

Conversely, suppose that item t_1 is allocated to the second agent, and the second item arrives in correspondence with instance \mathcal{I}_2 . The deterministic algorithm then achieves a NSW of $\sqrt{\epsilon(c+1)}$. For ϵ sufficiently small and c large enough however, the optimal allocation would be to allocate t_1 to the first agent and the remaining to the second, which yields a NSW of $\sqrt{(\epsilon+1)c}$. The corresponding competitive ratio is thus $\sqrt{\frac{(\epsilon+1)c}{\epsilon(c+1)}}$. For any given $M > 1$, we take $c = 4M^2$ and $\epsilon = \frac{1}{4M^2}$, to obtain

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} = \sqrt{(1 + 4M^2) \cdot \frac{4M^2}{1 + 4M^2}} = 2M > M.$$

This implies that the NSW of the algorithm can be arbitrarily large, and we finish the proof. \blacktriangle

We note that this is in stark contrast to the identical setting where the simple greedy algorithm guarantees a constant competitive ratio in the identical setting. Such a negative result stems from the fact that a zero-valued item may induce non-zero utility for only a subset of the agents, an added complexity of the affine utility setting that depends on the b_i terms and minimum item value.

This motivates us to parameterize the achievable competitive ratio with respect to $(b_i)_{i \in [n]}$ and the instance minimum item value. Let $v = \min_{t \in [T]} u(\{t\})$ be the minimum item value and

$b = \min_{i \in [n]} b_i$. We additionally define

$$B = \left(\prod_{i \in [n]} \frac{v + b_i}{v + b} \right)^{1/n} \quad (4.1)$$

The following theorem presents that the competitive ratio is upper bounded by $e^{1/e}$ times B under the affine utility setting.

Theorem 13. *In the affine value setting, there exists an algorithm with a competitive ratio at most $Be^{1/e}$.*

Proof. By Theorem 11, it suffices to consider cases where $a_i = 1$ for all $i \in [n]$. Suppose that each agent i is equipped with parameters $(1, b_i)$ for $i \in [n]$. We consider a greedy-like algorithm in which we allocates the incoming item to the least satisfied agent, but by assuming that each agent i is equipped with parameters $(1, b)$, not $(1, b_i)$ for $i \in [n]$. Denote this algorithm by ALG_1 . Given a problem instance \mathcal{I}_1 where the agent parameters are given by $(1, b_i)_{i \in [n]}$, denote by $\text{NSW}_{\text{ALG}_1, \mathcal{I}_1}$ the NSW of the allocation under ALG_1 and $\text{NSW}_{\text{OPT}, \mathcal{I}_1}$ the optimal offline NSW. Now consider a problem instance \mathcal{I}_2 such that the agent parameters are given by $(1, B)$ for each agent. Denote by ALG_2 the greedy algorithm which allocates the item to the agent with maximal valuation. It is obvious to see that $\text{NSW}_{\text{ALG}_1, \mathcal{I}_1} \geq \text{NSW}_{\text{ALG}_2, \mathcal{I}_2}$. Moreover, since \mathcal{I}_2 actually belongs to the identical setting as $b_i = B$ for all $i \in [n]$, by Theorem 5 we obtain $\text{NSW}_{\text{ALG}_2, \mathcal{I}_2} \cdot e^{1/e} \geq \text{NSW}_{\text{OPT}, \mathcal{I}_2}$. Suppose that the base utility for each item in \mathcal{I}_1 and \mathcal{I}_2 is given by x_t for $t \in [T]$. Then, consider a problem instance \mathcal{I}_3 such that all the base value x_t is scaled-up to be $x_t \left(\frac{v+b_i}{v+b} \right)$ for $t \in [T]$. Obviously, we have $\text{NSW}_{\text{OPT}, \mathcal{I}_3} \geq \text{NSW}_{\text{OPT}, \mathcal{I}_1}$ since each item at round t has larger (or equal) value in \mathcal{I}_3 for all $t \in [T]$. In addition, we observe that $\text{NSW}_{\text{OPT}, \mathcal{I}_3} = \text{NSW}_{\text{OPT}, \mathcal{I}_2} \left(\prod_{i \in [n]} \frac{v+b_i}{v+b} \right)^{1/n} = \text{NSW}_{\text{OPT}, \mathcal{I}_2} B$ since all the items are multiplied by the same constant factor. Combining the results we obtain

$$\begin{aligned} \text{NSW}_{\text{ALG}_1, \mathcal{I}_1} &\geq \text{NSW}_{\text{ALG}_2, \mathcal{I}_2} \geq \frac{1}{e^{1/e}} \text{NSW}_{\text{OPT}, \mathcal{I}_2} \\ &= \frac{1}{Be^{1/e}} \text{NSW}_{\text{OPT}, \mathcal{I}_3} \\ &\geq \frac{1}{Be^{1/e}} \text{NSW}_{\text{OPT}, \mathcal{I}_1}, \end{aligned}$$

and it completes the proof. ▲

Note that plugging $b_i = b$ yields the upper bound of $e^{1/e}$, which essentially coincides with the bound we provide in the identical setting. If v is comparably larger than b_i , then we still approximately have an upper bound of $e^{1/e}$. In addition, if b_i for $i \in [n]$ is bounded within a constant factor ρ from b , then one can easily observe that B is bounded above by ρ , which implies that the resulting competitive ratio is bounded above by $\rho e^{1/e}$.

5. Bivalue setting

Another natural reduction is to reduce the possibilities of the valuations that an agent can give to an item. Imagine that agents only tell if they like or dislike each item. In this setting we would have that $v_i(\{t\}) \in \{0, 1\}$ for all agents i and item t (note that from the scaling invariance it is the same than considering $v_i(\{t\}) \in \{0, r\}$ for any $r \in \mathbb{R}^+$). We call this setting **binary** for obvious reasons.

This setting has an easy interpretation: agents either like or dislike and want to maximize the number of liked items received, not caring about receiving or not disliked items.

In this setting we have already seen in Theorem 3 that we have an exponentially large lower bound on the competitive ratio depending on the number of agents. Nevertheless we will show that in when a large number of items are available we can get a positive result that is tight asymptotically, following the path of [Hajiaghayi et al. \[2022\]](#).

Then we will impose that $v_i(\{t\}) \in \{1, m\}$ for $m > 1$ (for the same reason than before this is equivalent to $v_i(\{t\}) \in \{a, b\}$ for $a, b \in \mathbb{R}^+$). Note that we expect this to be simpler than the binary setting as every agent values positively every item. We also have that the binary setting can be obtained when $m \rightarrow \infty$. We will call this setting m -bivalue setting. Here we will focus on a nice connection with the maximum matching problem that will allow us to derive some lower bounds on the competitive ratio.

This case diverges from the last one in the sense that the agents are happy about receiving disliked items, but value them less than the liked items.

5.1 Hardness in the offline setting

The bivalue problem has been studied in the offline setting. In the simpler setting of binary valuations [Babaioff et al. \[2020\]](#) designed an algorithm that runs in polynomial time and maximizes the NSW. Note that this is much simpler than the general setting that is APX -hard as discussed in the introduction.

In the other hand in the offline m -bivalue setting [Akrami et al. \[2021\]](#) showed that the problem is NP -hard and APX -hard. They provide a lower bound on the approximation factor and an algorithm that reaches a 1.0345-approximation polynomial algorithm. This algorithm finds the optimal allocation when $m \in \mathbb{N}$.

5.2 Binary setting

The exponential bounds when not enough items are present in the binary setting have already been discussed in Chapter 2. Because of this we only look at the setting where a *sufficiently large number*

of items arrive, as discussed in the following section.

5.2.1 Large number of items

The exponential lower bound exploits the fact that the NSW is zero if an agent does not get any positive valued item. In order to overcome this intrinsic hardness, we consider the binary setting with *sufficiently many* items – a canonical assumption in the literature (Hajiaghayi *et al.* [2022]). Specifically, we show in the following theorem that for a sufficiently large number of items of positive value for each agent, we can retain a linear competitive ratio bound.

Theorem 14. *In the binary value setting, if every agent values positively at least n items the greedy algorithm has a competitive ratio of at most $\Theta(n)$.*

Proof. Our main result essentially builds upon the following lemma.

Lemma 12. *If an agent positively values k items the greedy algorithm allocates him at least $\lfloor \frac{k}{n} \rfloor$ items.*

Proof of the lemma. Define $q := \lfloor k/n \rfloor$. We have to prove that after an agent has seen qn items he has received at least q . By the greedy algorithm if the agent has received r items all other agents will have received at most $r + 1$ of those items. Then $qn \leq r + (n - 1)(r + 1)$. Rearranging the inequality, this is equivalent to $r \geq (q - 1) + \frac{1}{n}$. This further implies that $r \geq q$, since r is an integer. ▲

Let k_i the number of positively valued items for agent i . From the lemma above, the competitive ratio will be at most

$$\left(\prod_{i \in [n]} \frac{k_i}{\lfloor \frac{k_i}{n} \rfloor} \right)^{1/n} \leq \left(\prod_{i \in [n]} 2n \right)^{1/n} = 2n,$$

and it finishes the proof. ▲

We can use the previous Lemma to improve a result in Hajiaghayi *et al.* [2022]. They deal about the called online Santa Claus problem. The only difference with our problem is that the objective is the maximization of the minimum of the values assigned to an agent, instead of the geometric mean. The following corollary improves the condition stated in the original paper from $n \log n$ to n and improves the dependency on the approximation factor.

Corollary 3 (Improvement over Theorem A.1. in Hajiaghayi *et al.* [2022]). *In the adversarial setting, there is an algorithm for the online Santa Claus problem that has a competitive ratio of $\frac{(q+1)n}{q}$ if $OPT \geq qn$, where $q := \lfloor k/n \rfloor$.*

Proof. The proof directly follows from Lemma 12 using the greedy algorithm. ▲

Remark 3. We note that our construction of the worst-case problem instance that gives us the exponential lower bound also improves Theorem 1.3 in Hajiaghayi et al. [2022]. More formally, it states that no online algorithm can obtain a competitive ratio better than $O(n)$, whereas we present an exponential bound.

We now show that the previous theorem is asymptotically tight, this is that the lower bound matches the $\Theta(n)$ upper bound. This shows that up to a constant the greedy algorithm is optimal.

Theorem 15. In the binary value setting, for any algorithm and any arbitrary function $f : \mathbb{N} \rightarrow \mathbb{N}$, the competitive ratio is at least $\Theta(n)$ even if we assume that every agent values positively at least $f(n)$ items.

Proof. Consider an input of n rounds where, in round i , a set of $(n - i + 1)k_i$ items valued 1 by agents $\sigma(1), \dots, \sigma(n - i + 1)$ arrive, where σ is a random permutation of the agents which we will consider to be the identity (without loss of generality) and the value of k_i will be set later. We invoke the following lemma on the optimal allocation for round i specified above.

Lemma 13. It is optimal to allocate the items evenly between the agents $1 \dots n - t + 1$ at round t . Allocating items evenly between the agents $1 \dots n - t + 1$ for every round t gives a $(1 + \epsilon)$ approximation of the optimal CR if the ratio k_{t+1}/k_t is big enough for all t .

Before proving this lemma, we show how it is leveraged to verify the theorem. Given this allocation we do not care about the $(1 + \epsilon)$ factor since we are dealing with asymptotically behaviour. Then we have that

$$\text{NSW}_{\text{ALG}} = \left(\prod_{i=1}^n \sum_{j=1}^i k_j \right)^{1/n}$$

and for $k_{i+1}(n - i) > k_i(n - i + 1)$ we have that

$$\text{NSW}_{\text{OPT}} = \left(\prod_{i=1}^n (n - i + 1)k_i \right)^{1/n}.$$

Given $\epsilon > 0$, suppose that $k_i > \frac{\sum_{j=1}^{i-1} k_j}{\epsilon}$. This further implies that $\frac{(n-i+1)k_i}{\sum_{j=1}^i k_j} > \frac{(n-i+1)}{(1+\epsilon)}$ and combining these inequalities yields the competitive ratio bound of $\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} > \frac{(n!)^{1/n}}{(1+\epsilon)} \in \Theta(n)$.

We now prove the lemma and complete the result.

Proof of Lemma 13. We will prove the lemma by induction. For $t = 0$ it is vacuously true. Suppose that after round $t - 1$ we have assign the same number of items to each agent in $[n - t + 1]$ and let this number be I . We have to see that it is optimal to assign the same number of items in round i to all of them. Call $K := (n - t + 1)k_{n-t+1}$. We can focus on $P(i_1, \dots, i_{t-1})$ the probability that we allocate i_j items to agent j for $1 \leq j \leq t - 1$ and $K - \sum_{j \leq t-1} i_j$ to agent t .

Assume that $K < \epsilon k_{n-t+2}$ for an small $\epsilon > 0$. Any agent who receives items this round will get at most K items and will be able to take *at least* k_{n-t+2} in the subsequent round. Furthermore, we have that

$$k_{n-t+2}(1 + \epsilon) > k_{n-t+2} + K$$

. Therefore, any agent that will receive items in later rounds can ignore the items received in this round and only be off by $(1 + \epsilon)$ to his final allocated value. This induces a $(1 + \epsilon)$ approximation on the ultimate NSW and is thus negligible in the asymptotic behavior of the competitive ratio for a fixed ϵ . We therefore focus on the items received by an agent in this round that will *not* receive any in the subsequent. Probabilistically, these agents are all identical so we seek to maximize

$$\sum_{i_1 + \dots + i_t = k} P(i_1, \dots, i_{t-1}) \frac{1}{t} \sum_{j \leq t-1} (I + i_j)^{1/n}.$$

The critical point occurs when $P(i_1, \dots, i_{t-1})$ is identically 1 for the agents that maximize the inner summation and 0 for the remainder. Lastly, taking partial derivatives of the optimization function, we have

$$\frac{1}{n} (I + i_j)^{1/n-1} = \frac{1}{n} \left(I + K - \sum_{l \leq t-1} i_l \right)^{1/n-1}$$

for all $j \leq t-1$ and so $K - \sum_{l \leq t-1} i_l = i_j \implies i_j = K/t = k_{n-t+1}$, concluding the proof. ▲

We can finally conclude the proof of Theorem 15 by bounding $k_1 > f(n)$. ▲

Finally seeing that in the online and offline adversarial models the CR matches with the ones in [Hajiaghayi et al. \[2022\]](#) problem we conjecture that it also matches in the random order model. It is stated formally in the following conjecture:

Conjecture 1. *In the random order setting, for all $\epsilon > 0$ if we receive more than $\Omega(\frac{\log n}{\epsilon^2})$ positively valued items for each agent, there exists an algorithm with $CR = (1 + \epsilon)$.*

5.3 m -Bivalue setting

In this setting we are interested in seeing how the competitive ratio varies when we parameterize the problem by m . When $m = 1$ the competitive ratio is 1 and from the binary setting we know that the competitive ratio arbitrarily big when we make m big enough. We begin by given a polynomial lower and upper ratio on the competitive ratio depending on m .

Theorem 16. *In the m -bivalue setting under an adversarial input, any online algorithm has a competitive ratio of at least $m^{5/18}$.*

Proof. For some integer k , consider a problem instance with $n = 3k$ agents and $3k$ items such that k items have value 1 for all agents (denote this by type A) and $2k$ items have value m for only one

agent (this agent can be different for each item, but each agent will at most value m one item) and value 1 for the rest of the agents (denote this by type B). For each type B item, the agent who has value m is chosen randomly and we assume that one agent values m at most one item yielding $\text{NSW}_{\text{OPT}} = (1^{2k}m^k)^{1/(3k)} = m^{2/3}$.

Now we consider a problem instance where the items are grouped into k bundles comprised of a type A and two type B with the former item arriving first. Each bundle arrives in sequential manner, ie. one after another. Suppose now that we inform the algorithm of the two agents who value two type B items by m and an agent that values all the items by 1 before each batch arrives. Since we are only adding information to the algorithm, this informed algorithm must have a competitive ratio at most that of any fully online algorithm. The informed algorithm will thus properly assign the last two items with probability $1/3$, assign only one properly with probability $1/2$, and mistakenly allocate with the remaining probability. As a result, the expected contribution to the geometric mean for the items in this arriving bundle will be at most $(1/6 + (1/2)m^{1/n} + (1/3)m^{2/n})$. Given that each bundles is independent from each other we can use that the multiplication of expected value is the expected value of the multiplication for independent random variables. This implies the bound $\text{NSW}_{\text{ALG}} \leq (1/6 + 1/2m^{1/n} + 1/3m^{2/n})^{n/3}$. Now taking $n \rightarrow \infty$ we obtain the bound of $\text{NSW}_{\text{ALG}} \leq m^{7/18}$, and thus the competitive ratio is at least $\frac{m^{2/3}}{m^{7/18}} = m^{5/18}$. \blacktriangle

After this lower bound we see that the naive round-robin algorithm gets competitive ratio of m .

Theorem 17. *In the m -bivalued setting, there is an algorithm with a competitive ratio less than or equal to m .*

Proof. Let $T = kn + q$ be the number of items where n is the number of agents, $q < n$ and $k > 0$. If we allocate all items equally to all agents, $n - q$ will receive k items and q will receive $k + 1$. Given the m -bivalued instance, the optimal offline NSW is upper bounded by the instance where all agents are allocated an item of value m . In the converse, the worst case online algorithm will allocate each item to the agents of value 1 for the arriving item. We thus have the competitive ratio bound

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} \leq \left(\frac{(km)^{n-q}((k+1)m)^q}{(k)^{n-q}((k+1))^q} \right)^{1/n} = m.$$

\blacktriangle

Remark 4. *It is straightforward to extend Theorem 17 to the setting in which all the items have value in $[x, xm]$ for some $x > 0$, instead of $\{1, m\}$.*

Given this result and the good results that the greedy algorithm achieves in the equal valuations setting and in the binary setting with a large number of items we might conjecture that the greedy gets a good performance in this setting as-well. This is not the case as its performance is arbitrarily close to the round-robin algorithm as stated in the following theorem.

Theorem 18. *Given any rational number m and n agents, in the m -bivalued setting, the greedy algorithm 1 has a competitive ratio of at least $m(1 - 1/n) + 1/n$.*

Proof. Suppose that $m = p/q \in \mathbb{Q}$ and let v_i be an item valued m by agent i and 1 by the rest. Consider the set of items valued: $\underbrace{v_1, \dots, v_1}_{p(n-1)+q}, \underbrace{v_2, \dots, v_2}_{p(n-1)+q}, \dots, \underbrace{v_n, \dots, v_n}_{p(n-1)+q}$. Divide the input into n rounds based on the type of item that we are receiving. We will prove by induction that, after each round, every agent must have the same allocated value. This is true for $i = 0$. Assume this is true until round i . In the following round, the agent $i + 1$ will receive q items augmenting his value by $\frac{p}{q} \cdot q = p$ and every other agent will receive p items augmenting their allocated values by $1 \cdot p = p$. Thus, the greedy procedure will induce an allocation resulting in $\text{NSW}_{\text{ALG}} = pn$. The optimal offline allocation will give all the items in round i to the corresponding agent, yielding $\text{NSW}_{\text{OPT}} = (p(n-1) + q)m$. Therefore, our competitive ratio is given by

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} = \frac{(p(n-1) + q)m}{pn} = m(1 - 1/n) + 1/n,$$

which verifies the lower bound. ▲

After this negative result with the greedy algorithm we will try to focus on a wider class of algorithms. For deriving more general lower bounds we will look at the scenario where the number of agents equals the number of items. This case was enough for getting the strong negative results in the binary case. We will draw inspiration from the online bipartite maximum matching problem so the next section focuses on a brief introduction to the problem.

5.3.1 Online bipartite maximum matching

A bipartite graph $G = (V, E)$ is one such we can partition $V = L \sqcup R$ such that all edges are between an vertex in L and a vertex in R (a condition equivalent to being 2-colorable). A matching of a graph is a subset of edges $M \subseteq E$ such that for all $u, v \in M$ we have $u \cap v = \emptyset$, i.e. there is no vertex that with two edges. A maximum matching of G is a matching of maximum cardinality along all matchings of G . Finding a maximum matching in a bipartite graph can be easily done with a max-flow algorithm. As expected we are interested in the online setting. In this setting we know from the beginning the vertices on L but the vertices in R appear one at the time with all their edges. One a vertex of R appears we can match it with an adjacent unmatched vertex of L . We will further assume that a maximum matching of size $n = |L|$ exists, i.e. all vertices on L can be matched at the same time.

Let's now consider the deterministic greedy algorithm for this problem, just assigning a unmatched adjacent vertex for the arriving vertex, breaking ties by a deterministic tie-breaking protocol. This algorithm has a competitive ratio of 2.

Lemma 14. *In the online bipartite maximum matching problem the deterministic greedy algorithm achieves a tight competitive ratio of 2 for all even $n = |L|$.*

Proof. Lower bound: consider $n = 2$ and a graph with $L = \{1, 2\}$, $R = \{3, 4\}$ and $E = \{\{1, 3\}, \{2, 3\}, \{1, 4\}\}$. Suppose that when the tie-breaking protocol picks the edge $\{1, 3\}$ when

the vertex 3 appears (if it picks the other substitute edge $\{1, 4\}$ by edge $\{2, 4\}$). Then it will not match vertex 4 and we will get a $CR = 2/1 = 2$. Copying this setting $n/2$ for any even n finish the lower bound proof.

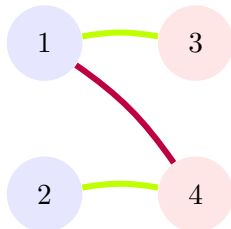


Figure 5.1: Graph that reaches the lower bound for $n = 2$. The maximum matching is formed by **lime** edges but the algorithm only selects the **purple** edge.

Upper bound: consider any maximum matching M and call the matching obtained by the algorithm A . For every edge $(u, v) = m \in M, u \in L, v \in R$ at least one of u or v appears in an edge of A . If that was not the case, when v arrived the algorithm would have picked m to be a part of A . Then for every edge in M we see a edge in A . Notice that we can see an edge in A by at most two edges in M , as M is a matching. Then $2|A| \geq M$ finishing the proof. ▲

One can think that randomizing the greedy might improve the algorithm or even that it the randomized greedy is the best algorithm. We will now see that neither of those holds. Consider the greedy algorithm with randomized tie breaking. We will call any random algorithm that only uses randomness for breaking ties a **random tie-breaker**, independently each time that it has to break a tie. Notice that in the lower bound example that we used in previous lemma this algorithm would get a competitive ratio of $4/3$. We will see that this can be worsened to 2 in the following lemma.

Lemma 15. *The randomized greedy has a competitive ratio of 2.*

Proof. Lower bound: For $n \in \mathbb{N}$ consider a graph $G = L \sqcup R$ with $|L| = |R| = n$ taking $L = [n]$. The i -th vertex of R to arrive will be adjacent to a vertex v of L if $v \geq i$ (see figure 5.3.1). If we run the randomized greedy and we call E_n the expected number of edges in the final matching we have that $E_n = 1 + \frac{1}{n}E_{n-1} + \frac{n-1}{n}E_{n-2}$, $E_0 = 0$. The 1 accounts for the edge picked when the first vertex arrive. If we select vertex i of L , with probability $1/n$, after this we will be in the same situation that we would be if we started with $n - 1$. If we pick one of the others we will not be able to use that one nor vertex 1, so we will be in the situation of starting with $n - 2$ with probability $1 - \frac{1}{n}$. Now we upper bound E_n .

Claim 7. $E_n \leq \frac{n}{2} + \frac{H_n}{2}$ where $H_n := \sum_{1 \leq i \leq n} \frac{1}{i}$ the n -th harmonic number.

Proof of Claim. For $n = 1$ we have $E_1 = 1 \leq 1/2 + 1/2$. Let the statement be true for all $i < n$. Then $E_n \leq 1 + \frac{1}{n}(\frac{n-1}{2} + \frac{H_{n-1}}{2}) + \frac{n-1}{n}(\frac{n-2}{2} + \frac{H_{n-2}}{2}) = \frac{n^2+1}{2n} + \frac{1}{n} \frac{H_{n-1}}{2} + \frac{n-1}{n} \frac{H_{n-2}}{2} \leq \frac{n}{2} + \frac{1}{2n} + \frac{H_{n-1}}{2} \leq \frac{n}{2} + \frac{H_n}{2}$. ▲

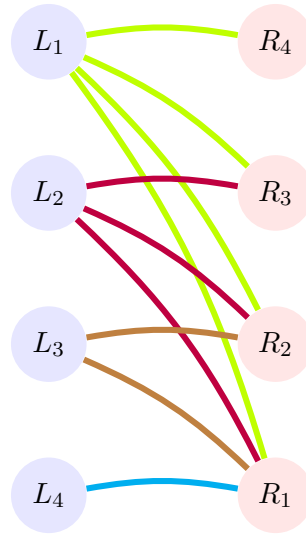


Figure 5.2: Graph used in the proofs of Lemma 15 and Theorem 19 for $n = 4$.

Using the claim, that $H_n \approx \log n$ and that the maximum matching has size n we have $CR_n \geq \frac{n}{n/2 + H_n/2} \approx 2 \frac{1}{1 + (\log n)/n} \rightarrow 2$ when $n \rightarrow \infty$.

Upper bound: Same proof as previous lemma. ▲

So randomizing the greedy is not enough. We have to correlate the random decisions taken by the algorithm to get a better algorithm. This is accomplished by the RANKING algorithm 3 proposed in the classical paper [Karp *et al.* \[1990\]](#).

This algorithm achieves a competitive ratio of $(1 - 1/e)^{-1} \approx 1.58$ better than the previously seen 2 competitive ratio. For doing so it randomly chooses a permutation of the vertices of the offline half of the graph and matches the arriving vertices using the order (or ranking) induced by the permutation. This implies that the decision in each step is deeply correlated with the previous decisions. A good proof of the competitive ratio of this algorithm can be found at [Birnbbaum and Mathieu \[2008\]](#).

Algorithm 3: Ranking algorithm

- 1 Randomly (and uniformly) choose a permutation σ of the vertices in L .
 - 2 When a vertex v of R arrives:
 - 3 Let $N(v)$ the neighbours of v that have not been matched.
 - 4 If $N(v) \neq \emptyset$ match v to the vertex u that minimizes $\sigma(u)$.
-

In the next section we show how this problem is connected to our problem in the bivalued setting and we also derive some bounds and ideas inspired by the exposed in this section.

5.3.2 Relation with bipartite maximum matching

We will restrict our problem to the case where exactly n items arrive. Then we have a nice relation with the bipartite maximum matching problem.

We can construct a bipartite in the following way: let the vertices on L (the non-online side) the n agents and the vertices in R the items that arrive online. An agent a and an item I are connected if the agent values the item with a value of m ($v_a(I) = m$).

Note that when an algorithm allocates items obtaining a non-zero NSW we get a matching in the previous graph: indeed each item can only be allocated to one agent and if an agent receives more than one item by the pigeonhole principle an item will have no allocated items inducing NSW = 0. More than that if the matching has size k the NSW will be $(m^k)^{1/n}$ so the objectives transforms into getting a large matching.

But there are some differences. Given a bipartite graph G , let M_{ALG}^G be the distribution of the size of the maximum matching, and K^G be the size of the offline maximum matching. Then, the online bipartite matching problem deals with $K^G/\mathbb{E}[M_{\text{ALG}}^G]$, this is it only cares about the expectancy of the maximum matching. In our problem we are more interested in the distribution of the sizes: Define $R = \inf_r \{r : [\mathbb{P}((K^G - M_{\text{ALG}}^G)/n \leq r) \geq c(r)], \forall G\}$, where $c(r) > 0$. Then, we can obtain the competitive ratio of

$$\frac{\text{NSW}_{\text{OPT}}}{\text{NSW}_{\text{ALG}}} = \frac{m^{K^G/n}}{m^{M_{\text{ALG}}^G/n}} \leq \frac{m^{K^G/n}}{c(R + \epsilon)m^{(K^G - n(R + \epsilon))/n}} = \frac{m^{R + \epsilon}}{c(R + \epsilon)},$$

for all $\epsilon > 0$ and for all problem instances that corresponds to the bipartite graph G . That said, we can still carry over the connection to $K^G/\mathbb{E}[M_{\text{ALG}}^G]$ as stated in the next lemma.

Lemma 16. *If $K^G/\mathbb{E}[M_{\text{ALG}}^G] \leq c$ then $R \geq 1 - 1/c$.*

Proof. We will use the same idea of Markov's inequality but for bounded by above random variables as seen in next claim.

Claim 8. *Let X be a random variable with $X \leq M$ a.e. and call $E = \mathbb{E}[X]$. Then for any $\lambda < E$ it holds $P(X > \lambda) \geq \frac{E - \lambda}{M - \lambda}$*

Proof of the claim.

$$\begin{aligned} E &= \int xP(X = x)dx = \int_{x > \lambda} xP(X = x)dx + \int_{x \leq \lambda} xP(X = x)dx \leq \\ &\leq M \int_{x > \lambda} P(X = x)dx + \lambda \int_{x \leq \lambda} P(X = x)dx = MP(X > \lambda) + \lambda P(X \leq \lambda) = \\ &= MP(X > \lambda) + \lambda(1 - P(X > \lambda)) \implies P(X > \lambda) \geq \frac{E - \lambda}{M - \lambda} \end{aligned}$$



Using this we want to see that for any $\epsilon > 0$ if we take $r = 1 - 1/c + \epsilon$ we have

$$\mathbb{P}((K^G - M_{\text{ALG}}^G)/n \leq r) \geq c(r) \iff \mathbb{P}(M_{\text{ALG}}^G \geq K^G - nr) \geq c(r)$$

Using the claim and the fact that M_{ALG}^G is bounded by K^G :

$$\begin{aligned} \mathbb{P}(M_{\text{ALG}}^G \geq K^G - nr) &\geq \frac{\mathbb{E}[M_{\text{ALG}}^G] - K^G + nr}{K^G - K^G + nr} \geq \frac{K^G/c - K^G + nr}{nr} = 1 - K^G \left(\frac{1 - 1/c}{nr} \right) \geq \\ &\geq 1 - \frac{1 - 1/c}{r} = \frac{\epsilon}{1 - 1/c + \epsilon} = c(r) > 0 \end{aligned}$$

▲

There is another difficulty. In the classical bipartite maximum matching if a vertex arrives and does not have a free neighbour we can just “pass”, don’t match it and go on with the next vertex. This is not the case in our version. We need to match every item to an agent that will not be able to receive more items. Then if we receive an item value 1 by all agents without allocated item we will need to pick an agent and allocate that item to it. This implies that the use of canonical greedy algorithm with deterministic tie-breaking in the online bipartite maximum matching will not give us a good competitive ratio. Consider two agents with two vertices, where the first has no neighbours (value m), and the second has only one neighbour, wherein any algorithm with deterministic tie-breaking can get a matching of size 0, in comparison with the usual online bipartite matching case where a deterministic greedy always get at least a matching of half the size of the maximum matching as we saw in the previous section.

5.3.3 Negative results inspired of random tie-breakers algorithms

We have seen that randomizing the greedy is not enough to derive a better algorithm for the online bipartite matching. Inspired in this and in the connection explained in the previous section we derive a very similar result for our problem for any random tie-breaker algorithm. Note that this algorithms are the ones used in almost all allocation problems.

Theorem 19. *In the m -bivalued setting under an adversarial input, any random tie-breaker algorithm has a competitive ratio of at least $m^{1/2-\epsilon}$ for all $\epsilon > 0$.*

Proof. Consider a set of items $[n]$ such that $v_i(a_j) = 1$ if $j < i$ and m otherwise (see figure 5.3.1). We denote $C(n, j)$ the probability of assigning to j agents an item valued m (and an item valued at 1 to the rest) in our described instance of n agents following a uniformly random tie-breaking algorithm. Define $A_n^\epsilon = \sum_{i=n(1+\epsilon)/2}^n C(n, i)$. We want to show that $A_n^\epsilon \rightarrow 0$ as $n \rightarrow \infty$, as this implies that the probability of having the NSW bigger than $m^{1/2+\epsilon}$ goes to 0.

Note that $C(1, 1) = 1$ and $C(n, j) = 0$ for $j < n/2$. We know derive a recurrence relation for this values:

Claim 9. $C(n, j) = \frac{1}{n}C(n-1, j-1) + \frac{n-1}{n}C(n-2, j-1)$.

Proof of the Claim 9. if we pick one agent we have two cases: if the agent is the one that only valued with m the first item we pass to the case $C(n-1, j-1)$ and this happens with probability $\frac{1}{n}$. In the other case we have to discard two agents as possible receivers of future m valued items: the one that we assigned the item to and the one that will not be offered any other item that he values m in the future. This happens with probability $1-1/n$ and moves us to the case $C(n-2, j-1)$. \blacktriangle

Rewrite $Q(n, k) = C(2n - k, n)$. Note that $n \geq k$ if $Q(n, k) > 0$. It suffices to upper bound the quantity $Q(n, k)$. From Claim 9, we obtain

$$\begin{aligned} Q(n, k) &= C(2n - k, n) \\ &= \frac{1}{2n - k} C(2n - k - 1, n - 1) \\ &\quad + \frac{2n - k - 1}{2n - k} C(2n - k - 2, n - 1) \\ &= \frac{1}{2n - k} Q(n - 1, k - 1) + \frac{2n - k - 1}{2n - k} Q(n - 1, k). \end{aligned}$$

Based on the recurrence relation above, we further use the following claim.

Claim 10. $Q(n, k) \leq \frac{1}{k!} \binom{n}{k}$.

Proof of the Claim 10. To compute the quantity $Q(n, k)$ based on the recurrence relation, in case $n = 1$, we need to compute for two cases such that $k = 0, 1$. Generally, to compute $Q(n, k)$ from $Q(n-1, k-1)$, we need to iterate recurrence relation k times, since the other term of the recurrence does not change k . We can then upper bound the value of $Q(n, k)$ counting all the possible orders of taking the steps of the form $(n, k) \rightarrow (n-1, k)$ and $(n, k) \rightarrow (n-1, k-1)$ until we reach $(0, 0)$. A classical combinatoric's argument states that there are $\binom{n}{k}$ of doing so. We can then multiply this number with the path that has the highest probability. It will be less than $\frac{1}{k!}$ since for taking the step $(n, k) \rightarrow (n, k-1)$ we will have probability $\frac{1}{2n-k} \leq \frac{1}{k} \iff k \leq n$. \blacktriangle

This implies that

$$C(n, k) = Q(k, 2k - n) \leq \binom{k}{2k - n} \frac{1}{(2k - n)!}.$$

From the definition of A_n^ϵ , we obtain

$$\begin{aligned}
A_n^\epsilon &\leq \sum_{k=n(1+\epsilon)} \binom{k}{2k-n} \frac{1}{(2k-n)!} \\
&\leq \frac{1}{(\lfloor \epsilon n \rfloor)!} \sum_{k=n(1+\epsilon)/2}^n \binom{k}{2k-n} \\
&\stackrel{(a)}{\leq} \frac{1}{(\lfloor \epsilon n \rfloor)!} \sum_{k=n(1+\epsilon)/2}^n \binom{k}{\lfloor k/2 \rfloor} \\
&\stackrel{(b)}{\leq} \frac{n}{(\lfloor \epsilon n \rfloor)!} \binom{n}{\lfloor n/2 \rfloor} \\
&\stackrel{(c)}{\leq} C \frac{n}{\sqrt{\epsilon n} (\epsilon n/e)^{\epsilon n}} \frac{2^n}{\sqrt{n}} \\
&\leq C' \frac{(2e)^n}{(\epsilon n)^{\epsilon n}},
\end{aligned}$$

where we used binomial inequalities in (a) and (b), Stirling's approximation in (c), and big enough C and C' . Therefore, we conclude that A_n^ϵ converges to zero as n goes to ∞ . This completes the proof as the probability of getting a $\text{NSW}_{\text{ALG}} > m^{1/2+\epsilon}$ goes to 0 and $\text{NSW}_{\text{OPT}} = m$ so $CR \leq m^{1/2-\epsilon}$. \blacktriangle

Remark 5. *The previous proof differs to the ones in the bipartite maximum matching because we needed to show a concentration of the distribution around the mean. This would have been easier if we showed that the variance of the distribution goes to 0. For doing so it is enough to define $X_i := 1$ if agent i is matched and 0 otherwise. Then we would be interested in the distribution of $Y = \frac{\sum_{i=1}^n X_i}{n}$ and it would be enough that the covariance of X_i and X_j is ≤ 0 using that $\text{Var}(Y) = \sum_{i=1}^n \text{Var}(X_i) + \sum_{i \neq j} \text{Cov}(X_i, X_j)$. But it might not be the case that the covariances are non-positive.*

All this hints that an algorithm similar to the already seen RANKING is needed for getting a better performance in this problem. It would likely get a competitive ratio of $m^{(1-1/e)^{-1}}$ as we have seen we expect the distribution being concentrated around the mean.

6. Conclusion

In this study, we have undertaken a comprehensive investigation of the online NSW maximization problem from various perspectives. Our primary focus has been on determining lower bounds for the competitive ratio, and we have discovered that it is impossible to achieve good algorithms under certain general conditions. However, by examining some natural restricted scenarios, we have been able to identify some positive results.

For instance, in the equal valuations scenario, we have found that the greedy algorithm performs quite well, with the lower and upper bounds being very close. Similarly, we have demonstrated that the greedy algorithm is asymptotically tight when we receive a large number of items with binary valuations. However, in the m -bivalued setting, we have been unable to find a good positive result, although we have derived lower bounds for a wide range of algorithms. In particular, we have shown that the greedy algorithm is not effective in this setting.

Given the variety of scenarios we have explored, there are numerous avenues for further research. For example, we could focus on closing the gap in the equal valuations setting, although this would require a sophisticated algorithm. Proving that the proposed randomized algorithm, or other, breaks the deterministic bound would be a big breakthrough. Alternatively, we could attempt to find a good algorithm for the bivalued setting, perhaps by drawing inspiration from the bipartite matching problem. Finally, we believe that the most promising direction for future research would be to investigate bounds and algorithms under the random order arrival model, which is the setting that best describes real-world situations and has received a lot of attention in similar problems in recent times. Although we have established some general lower bounds, it would be particularly interesting to explore the equal valuations and large number of items scenarios in this context.

Bibliography

- Hannaneh Akrami, Bhaskar Ray Chaudhury, Martin Hoefer, Kurt Mehlhorn, Marco Schmalhofer, Golnoosh Shahkarami, Giovanna Varricchio, Quentin Vermande, and Ernest van Wijland. Maximizing nash social welfare in 2-value instances, 2021.
- Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms (TALG)*, 2(4):640–660, 2006.
- Haris Aziz, Ioannis Caragiannis, Ayumi Igarashi, and Toby Walsh. Fair allocation of combinations of indivisible goods and chores, 2018.
- Moshe Babaioff, Tomer Ezra, and Uriel Feige. Fair and truthful mechanisms for dichotomous valuations, 2020.
- Siddhartha Banerjee, Vasilis Gkatzelis, Artur Gorokh, and Billy Jin. Online nash social welfare maximization with predictions. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–19. SIAM, 2022.
- Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC '06*, page 31–40, New York, NY, USA, 2006. Association for Computing Machinery.
- Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding fair and efficient allocations. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 557–574, 2018.
- S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. *Proceedings of the twenty-second annual ACM symposium on Theory of computing - STOC '90*, 1990.
- Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *ACM SIGACT News*, 39(1):80–87, 2008.
- Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research*, 34(2):270–286, 2009.
- Colin F. Camerer. *Behavioral game theory experiments in strategic interaction*. New Age International, 2014.

- Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. *ACM Transactions on Economics and Computation (TEAC)*, 7(3):1–32, 2019.
- Mithun Chakraborty, Erel Segal-Halevi, and Warut Suksompong. Weighted fairness notions for indivisible items revisited. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):4949–4956, Jun. 2022.
- Richard Cole and Vasilis Gkatzelis. Approximating the nash social welfare with indivisible items. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 371–380, 2015.
- Richard Cole, Nikhil Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V. Vazirani, and Sadra Yazdanbod. Convex program duality, fisher markets, and nash social welfare. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17*, page 459–460, New York, NY, USA, 2017. Association for Computing Machinery.
- Nikhil R Devanur and Kamal Jain. Online matching with concave returns. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 137–144, 2012.
- Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal–dual algorithm for a convex program. *J. ACM*, 55(5), nov 2008.
- MohammadTaghi Hajiaghayi, MohammadReza Khani, Debmalya Panigrahi, and Max Springer. Online algorithms for the santa claus problem. *NeurIPS*, 2022.
- Zhiyi Huang, Minming Li, Xinkai Shu, and Tianze Wei. Online nash welfare maximization without predictions, 2022.
- Mamoru Kaneko and Kenjiro Nakamura. The nash social welfare function. *Econometrica: Journal of the Econometric Society*, pages 423–435, 1979.
- Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.
- Richard M. Karp. An introduction to randomized algorithms. *Discrete Applied Mathematics*, 34(1):165–201, 1991.
- Euiwoong Lee. Apx-hardness of maximizing nash social welfare with indivisible items. *Information Processing Letters*, 122:17–20, 2017.
- Roos Magnus and Rothe Jörg. Complexity of social welfare optimization in multiagent resource allocation, 2010.

- Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.
- Aranyak Mehta et al. Online matching and ad allocation. *Foundations and Trends[®] in Theoretical Computer Science*, 8(4):265–368, 2013.
- Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*, 2018.
- Hervé Moulin. *Fair Division and Collective Welfare*. The MIT Press, 2003.
- Trung Thanh Nguyen, Magnus Roos, and Jörg Rothe. A survey of approximability and inapproximability results for social welfare optimization in multiagent resource allocation. *Annals of Mathematics and Artificial Intelligence*, 68(1-3):65–90, 2013.
- Benjamin Plaut and Tim Roughgarden. Almost envy-freeness with general valuations, 2017.

A. Code 1

The code of this appendix finds optimal distributions of utilities in the equal valuation settings as discussed in Section 3.4 and generates the figures 3.1. It uses an iterative numeric algorithm that converges to the solution proved to be optimal in the aforementioned section.

```
1  # finding optimal shapes for the NSW online equal valuations problem
2  import math
3  import matplotlib.pyplot as plt
4  import random as rnd
5  import numpy as np
6
7  EPOCHS = 40
8  BATCH_SIZE = 1000
9  INITIAL_EPS = 0.3
10 DIV_EPS = 2
11
12 MIN_N = 50
13 MAX_N = 60
14 N_STEP = 10
15
16
17 def optimal(j):
18     return math.pow(1/j, j)
19
20 def printer(n, ratios, shape):
21     x_ax = [i for i in range(1, n+1)]
22     inv_shape = [1/x for x in shape]
23
24     fig, ax1 = plt.subplots()
25
26     color = 'tab:red'
27     ax1.set_xlabel('Item index (i)')
28     ax1.set_ylabel('c', color=color)
29     ax1.plot(x_ax, ratios, color=color)
30     ax1.tick_params(axis='y', labelcolor=color)
31     x_label = [1] + [i for i in range(10, n+1, 10)]
```



```

32     ax1.set_xticks(x_label, x_label)
33
34     ax2 = ax1.twinx()
35
36     color = 'tab:blue'
37     ax2.set_ylabel(r'$a_i$', color=color)
38     ax2.plot(x_ax, shape, color=color)
39     ax2.tick_params(axis='y', labelcolor=color)
40
41     locs = ax2.get_yticks()
42     locs = locs[1:-1]
43     y_label_text = [(f"%0.2f"%(i*n))+"/n" for i in locs]
44     ax2.set_yticks(locs, y_label_text)
45
46     plt.axvline(n/math.exp(math.e-1), color='purple', ls='--',
47     → label=r'$l=\frac{n}{e^{e-1}}$')
48
49     fig.tight_layout()
50     plt.legend(loc="lower right")
51     plt.savefig('n' + str(n) + '_double_curve.png', dpi=300)
52
53     fig, ax1 = plt.subplots()
54     color = 'tab:red'
55     interesting_line = [math.e * i for i in range(2, n)]
56
57     plt.axhline(n/math.exp(math.e-2), color='purple', ls='--',
58     → label=r'$1/a_1=\frac{n}{e^{e-2}}$')
59     ax1.plot(list(range(2, n)), interesting_line, color="green", ls='--',
60     → label=r'$a_i = ei$')
61
62     ax1.set_xlabel('Item index (i)')
63     ax1.set_ylabel(r'$1/a_i$')
64     ax1.plot(x_ax, inv_shape, label="1/a_i")
65
66     locs = ax1.get_yticks()
67     locs = locs[2:-1]
68     y_label_text = [(f"n/"+f"%0.2f"%(n/i)) for i in locs]
69     ax1.set_yticks(locs, y_label_text)

```

```

67     x_label = [1] + [i for i in range(10, n+1, 10)]
68     ax1.set_xticks(x_label, x_label)
69
70
71     plt.legend(loc="upper left")
72     plt.savefig('n' + str(n)+ '_inverse_a.png', dpi=300)
73
74 for n in range(MIN_N, MAX_N, N_STEP):
75     eps = INITIAL_EPS
76     ratio_inv = 0
77
78     shape = [rnd.random() for _ in range(n)]
79     S = sum(shape)
80     shape = [x/S for x in shape]
81     shape.sort(reverse=True)
82
83     for _ in range(EPOCHS):
84         for _ in range(BATCH_SIZE):
85             min_ratio = shape[0]
86             worst_pos = 0
87             prod = shape[0]
88             inv_ratios = [shape[0]]
89             for i in range(1, n):
90                 prod *= shape[i]
91                 ratio_act = prod/optimal(i+1)
92                 if ratio_act < min_ratio:
93                     min_ratio = ratio_act
94                     worst_pos = i
95                 inv_ratios.append(ratio_act)
96             ratio_inv = min_ratio
97             shape[worst_pos] += eps
98             shape = [x-eps/n for x in shape]
99             shape.sort(reverse=True)
100         eps /= DIV_EPS
101
102     ratios = [pow(1/x, 1/n) for x in inv_ratios]
103     printer(n, ratios, shape)
104

```

```
105     print(n)
106     print("c:", math.pow(1/ratio_inv, 1/n))
107     flat_end = 0
108     for i in range(n-1):
109         if abs(1/shape[i]-1/shape[i+1]) < 0.5*math.e/n:
110             flat_end = i
111     print("flat end:", flat_end, flat_end/n)
112
113     inv_shape = [1/x for x in shape]
114
115     print("initial constant", n/inv_shape[0], "slope",
116           ↪ inv_shape[-1]-inv_shape[-2])
117     print("\n")
```

B. Code 2

Program that creates and solves the linear system needed in the proof of Theorem 10. Check the Theorem proof for more information.

```
1 import numpy as np
2 import math
3 from scipy.optimize import linprog
4 from matplotlib import pyplot as plt
5
6 MIN_N = 300
7 MAX_N = 1500
8 STEP_N = 300
9
10 for n in range(MIN_N, MAX_N, STEP_N):
11     obj = [0 for _ in range(n)]
12     # -1 because we are maxizing p1, see Theorem for more info
13     obj[0] = -1
14     mat = [[0 for _ in range(n)] for _ in range(n)]
15     for number_big_items in range(n):
16         for agents_with_alloc_items in range(1,n+1):
17             # if we allocated items to too few agents some agent
18             # ↪ will be left
19             # without items and the NSW will be 0
20             if agents_with_alloc_items+number_big_items < n:
21                 mat[number_big_items][agents_with_alloc_items-1]
22                 ↪ = 0
23                 continue
24             # we are only interested in the agents without a big
25             # ↪ item
26             used_agents = n-number_big_items
27             # all agents with allocated goods have at least one
28             # ↪ good
29             distributed_goods = n-agents_with_alloc_items
30             # base = how many items will at least receive the
31             # ↪ agents without a big item
32             base = (n-agents_with_alloc_items)//used_agents +1
```

```

28         # but some will receive an extra item
29         extra = (n-agents_with_alloc_items) %
           ↪ used_agents
30         nsw = pow(base+1, extra) * pow(base,
           ↪ used_agents-extra)
31         nsw = pow(nsw, 1/n)
32         mat[number_big_items][agents_with_alloc_items-1] = nsw
33     best_nsw = [max(v) for v in mat]
34
35     # Restrictions on CR, see theorem proof.
36     # p_1 <= E[NSW(number of big items)]/max(NSW(number of big items)) for
           ↪ all number of big items.
37     lhs = [[1]+[0]*(n-1) for _ in range(n-1)]
38     for number_big_items in range(1,n):
39         for agents_with_alloc_items in range(1,n+1):
40             lhs[number_big_items-1][n-agents_with_alloc_items]
           ↪ -=mat[number_big_items][agents_with_alloc_items-1]
41                 /best_nsw[number_big_items]
42     lhs_ineq = np.array(lhs)
43     rhs_ineq = [0 for _ in range(n-1)]
44
45     # sum of probabilities = 1
46     lhs_eq = [[1 for _ in range(n)]]
47     rhs_eq = [1]
48     # probabilities between 0 and 1
49     bnd = [(0, 1) for _ in range(n)]
50
51     opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq,
52                 A_eq=lhs_eq, b_eq=rhs_eq, bounds=bnd,
53                 method="revised simplex")
54     print(-1/opt["fun"])

```

C. Code 3

This is a simulator and competitive ratio calculator for the family of algorithms described in Section 3.5.2. The `cond_function` has to be implemented for the desired algorithm within the family. Running the program with option `-h` explains how to use it.

```
1  import argparse
2  from scipy.optimize import linprog
3  import math
4  import numpy.random as rand
5  import numpy as np
6  import matplotlib.pyplot as plt
7  import ast
8  import os
9
10 INFINITY = 1e9
11 EPS = 1e-7
12
13 parser = argparse.ArgumentParser()
14 parser.add_argument("--min_agents",    default=2,    type=int,
15     ↪ help="Minimum number of agents used.")
16 parser.add_argument("--max_agents",    default=2,    type=int,
17     ↪ help="Maximum number of agents used.")
18 parser.add_argument("--step_agents",   default=1,    type=int,
19     ↪ help="Stride of iterated agents.")
20 parser.add_argument('--tight_cases',   default=False, action='store_true',
21     ↪ help='Print the tight cases of the linear optimization.')
22 parser.add_argument('--random_cases',  default=0,    type=int,
23     ↪ help='Number of random cases to generate.')
24 parser.add_argument('--perms',        default=1,    type=int,
25     ↪ help='Number of permutations of each case.')
26 parser.add_argument('--approximation', default=False, action='store_true',
27     ↪ help='Use an approximation for calculating difficult optimal values.')
28 parser.add_argument('--generate',     default=False, action='store_true',
29     ↪ help='Generate new cases and store them.')
30 parser.add_argument('--file',         default=None, type=str,
31     ↪ help='Path to file where store/read cases.')
```

```
23
24 rand.seed(42)
25
26 def main(args):
27     if args.generate:
28         if os.path.exists(args.file):
29             os.remove(args.file)
30         for num_agents in range(args.min_agents, args.max_agents+1,
31             ↪ args.step_agents):
32             save_cases(num_agents, args.file, args)
33             print("generated cases for #agents = " + str(num_agents))
34
35     all_cases = None
36     if args.file is not None:
37         all_cases = read_cases(args.file)
38
39     max_cr = 0
40     crs = []
41     dists = []
42     greedy_crs = []
43     for num_agents in range(args.min_agents, args.max_agents+1,
44         ↪ args.step_agents):
45         cases = None
46         opts = None
47         if all_cases is not None:
48             cases = all_cases[num_agents][0]
49             opts = all_cases[num_agents][1]
50         cr, dist, tight_cases, cr_greedy =
51         ↪ opt_dist_and_cr_of_base_cases(num_agents, cases, opts, args)
52         crs.append(cr)
53         greedy_crs.append(cr_greedy)
54
55     dists.append((num_agents, dist))
56     max_cr = max(cr, max_cr)
57     print("num agents ", num_agents)
58     print("cr:         ", cr)
59     print("max cr:      ", max_cr)
60     print("cr greedy:   ", cr_greedy)
```

```
58     print("dist:      ", dist)
59     if args.tight_cases:
60         print_tights(tight_cases)
61     print("-"*20)
62
63     print(crs)
64     print(greedy_crs)
65
66 def cond_function(s, b, n, D):
67     return None
68 def print_tights(tights):
69     print("TIGHT CASES")
70     for x in tights:
71         out = ""
72         for y in x:
73             out += "INF" if y >= INFINITY else str(int(y))
74             out += " "
75         print(out)
76
77 def iteration(item, agents, D):
78     B = agents[:D]
79     S = agents[D:]
80     b = sum(B)
81     s = sum(S)
82     n = len(agents)
83     if D == n or item >= cond_function(s, b, n, D):
84         agents[-1] += item
85     else:
86         agents[D-1] += item
87     agents.sort(reverse=True)
88     return agents
89
90 def nsw_corr_with_D(items, num_agents, D):
91     agents = [0 for _ in range(num_agents)]
92     for item in items:
93         agents = iteration(item, agents, D)
94     return nsw(agents)
95
```



```
96 def nsw(agents):
97     mult = 1
98     for x in agents:
99         mult *= pow(x, 1/len(agents))
100     return mult
101
102 def opt_nsw_rec(items, agents, pos_items):
103     if pos_items == len(items):
104         return nsw(agents)
105     mx = 0
106     seen = set()
107     for i in range(len(agents)):
108         prev = agents[i]
109         if prev >= INFINITY or prev in seen:
110             continue
111         agents[i] += items[pos_items]
112         mx = max(mx, opt_nsw_rec(items, agents, pos_items+1))
113         agents[i] = prev
114         seen.add(prev)
115     return mx
116
117 def opt_nsw(items, num_agents):
118     items2 = list(items)
119     items2.sort(reverse=True)
120     return opt_nsw_rec(items, [0 for _ in range(num_agents)], 0)
121
122 def insert_perms(case, opt, cases, opts, perms):
123     for _ in range(perms):
124         cases.append(case)
125         opts.append(opt)
126         case = rand.permutation(case)
127
128 def generate_base_cases(num_agents, args=None):
129     all_cases = []
130     opt_values = []
131     for small in range(10*num_agents+1):
132         for big in range(num_agents+1):
133             if big+small < num_agents:
```

```

134         continue
135
136     case = [1]*small + [INFINITY]*big
137     case2 = [INFINITY]*big+[1]*small
138
139     if big == num_agents:
140         insert_perms(case, INFINITY, all_cases, opt_values,
141             ↪ args.perms)
142         insert_perms(case2, INFINITY, all_cases, opt_values,
143             ↪ args.perms)
144     else:
145         extra_agents = num_agents-big
146         red_inf = pow(INFINITY, 1/num_agents)
147         base_items = small // extra_agents
148         base_plus_1_agents = small % extra_agents
149         base_agents = extra_agents-base_plus_1_agents
150         opt = pow(red_inf, big) * pow(base_items,
151             ↪ base_agents/num_agents) * pow(base_items+1,
152             ↪ base_plus_1_agents/num_agents)
153
154         insert_perms(case, opt, all_cases, opt_values, args.perms)
155         insert_perms(case2, opt, all_cases, opt_values, args.perms)
156
157 for small in range(num_agents+1):
158     for big in range(2,num_agents+1):
159         if big+small < num_agents:
160             continue
161         small_infinity = pow(INFINITY, 1/big)
162         bigs = [pow(small_infinity, i) for i in range(1, big+1)]
163         case = [1]*small + bigs
164         if big == num_agents:
165             bigs = [pow(x, 1/num_agents) for x in bigs]
166             opt = math.prod(bigs)
167             insert_perms(case, opt, all_cases, opt_values, args.perms)
168         else:
169             extra_agents = num_agents-big
170             base_items = small // extra_agents
171             base_plus_1_agents = small % extra_agents

```

```

168         base_agents = extra_agents-base_plus_1_agents
169         bigs = [pow(x, 1/num_agents) for x in bigs]
170         opt = math.prod(bigs) * pow(base_items,
171             ↪ base_agents/num_agents) * pow(base_items+1,
172             ↪ base_plus_1_agents/num_agents)
171         insert_perms(case, opt, all_cases, opt_values, args.perms)
172
173     for small in range(0):
174         for big in range(num_agents):
175             if big+small < num_agents:
176                 continue
177             case = [i for i in range(1, small+1)]+ [INFINITY]*big
178             opt_val = None
179             if args.approximation:
180                 suma = small*(small+1)//2
181                 opt_val = pow(suma/(num_agents-big),
182                 ↪ (num_agents-big)/num_agents)*pow(INFINITY, big/num_agents)
182             else:
183                 opt_val = opt_nsw(case, num_agents)
184             insert_perms(case, opt_val, all_cases, opt_values, args.perms)
185
186             case = [small+1-i for i in range(1, small+1)]+ [INFINITY]*big
187             insert_perms(case, opt_val, all_cases, opt_values, args.perms)
188
189     for small in range(0):
190         for big in range(num_agents):
191             if big+small < num_agents:
192                 continue
193             case = [2**i for i in range(0, small)]+ [INFINITY]*big
194             solos = num_agents-big-1
195             val_solos = [pow(2**i, 1/num_agents) for i in range(small-solos,
196                 ↪ small)]
196             opt_val = pow(INFINITY, big/num_agents) * math.prod(val_solos) *
197                 ↪ pow(2**(small-solos)-1, 1/num_agents)
197             insert_perms(case, opt_val, all_cases, opt_values, args.perms)
198
199             case = [2**(small-i) for i in range(0, small)]+ [INFINITY]*big
200             insert_perms(case, opt_val, all_cases, opt_values, args.perms)

```

```

201
202     for small in range(num_agents+1):
203         for mid in range(num_agents+1):
204             for big in range(num_agents+1):
205                 if big+small+mid < num_agents:
206                     continue
207                 if mid + big >= num_agents:
208                     continue
209
210                 extra_agents = num_agents-big-mid
211                 red_inf = pow(INFINITY, 1/num_agents)
212                 base_items = small // extra_agents
213                 base_plus_1_agents = small % extra_agents
214                 base_agents = extra_agents-base_plus_1_agents
215                 case = [1]*small+[math.sqrt(INFINITY)]*mid + [INFINITY]*big
216                 opt = pow(red_inf, big) * pow(math.sqrt(red_inf), mid) *
                ↪ pow(base_items, base_agents/num_agents) *
                ↪ pow(base_items+1, base_plus_1_agents/num_agents)
217                 insert_perms(case, opt, all_cases, opt_values, args.perms)
218                 case = [math.sqrt(INFINITY)]*mid + [1]*small+[INFINITY]*big
219                 insert_perms(case, opt, all_cases, opt_values, args.perms)
220
221     for _ in range(args.random_cases):
222         length = rand.randint(num_agents, 2*num_agents)
223         case = [rand.random() for _ in range(length)]
224         opt_val = opt_nsw(case, num_agents)
225         insert_perms(case, opt_val, all_cases, opt_values, args.perms)
226
227
228     return all_cases, opt_values
229
230 def save_cases(n, file=None, args=None):
231     f = open(file, "a")
232     cases, opt_values = generate_base_cases(n, args)
233     opts = [opt if opt is not None else opt_nsw(case, num_agents) for (case,
                ↪ opt) in zip(cases, opt_values)]
234     cases = [list(x) for x in cases]
235     f.write(str(n))

```

```

236     f.write("\n")
237     f.write(str(cases))
238     f.write("\n")
239     f.write(str(opts))
240     f.write("\n")
241     f.close()
242
243 def read_cases(file):
244     f = open(file, "r")
245     all_cases = {}
246     lines = f.readlines()
247     for (n, cases, opts) in zip(lines[0::3], lines[1::3], lines[2::3]):
248         n = int(n)
249         cases = ast.literal_eval(cases)
250         opts = ast.literal_eval(opts)
251         all_cases[n] = (cases, opts)
252     f.close()
253     return all_cases
254
255 def opt_dist_and_cr_of_base_cases(num_agents, cases=None, opts=None,
↪ args=None):
256     if cases == None:
257         cases, opt_values = generate_base_cases(num_agents, args)
258         opts = [opt if opt is not None else opt_nsw(case, num_agents) for
↪ (case, opt) in zip(cases, opt_values)]
259
260     nsw_to_dist = [[nsw_corr_with_D(case, num_agents, D) for D in
↪ range(1,num_agents+1)] for case in cases]
261     dist_cr = [[nsw/opt for nsw in nsws] for (opt, nsws) in zip(opts,
↪ nsw_to_dist)]
262
263     cr_greedy = 1
264     for x in dist_cr:
265         cr_greedy = min(cr_greedy, x[-1])
266     cr_greedy = 1/cr_greedy
267
268     lhs_ineq = [-cr for cr in crs]+[1] for crs in dist_cr]
269     rhs_ineq = [0 for _ in range(len(lhs_ineq))]

```

```
270
271 lhs_eq = [[1]*num_agents+[0]]
272 rhs_eq = [[1]]
273
274 obj = [[0]*num_agents+[-1]]
275 bnd = [(0, 1) for _ in range(num_agents+1)]
276
277 opt = linprog( c=obj,
278               A_ub=lhs_ineq, b_ub=rhs_ineq,
279               A_eq=lhs_eq,   b_eq=rhs_eq,
280               bounds=bnd,
281               method="highs")
282
283 tight_cases = []
284 for (slack, case) in zip(opt["slack"], cases):
285     if abs(slack) < EPS:
286         tight_cases.append(case)
287
288 cr = -1/opt["fun"]
289 dist = opt["x"][:-1]
290 return cr, dist, tight_cases, cr_greedy
291
292
293 if __name__ == "__main__":
294     args = parser.parse_args([] if "__file__" not in globals() else None)
295     main(args)
296
297
```