Max-Min Fair Allocation of Indivisible Goods

Daniel Golovin

June 2005
CMU-CS-05-144

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We consider the problem of fairly allocating a set of m indivisible goods to n agents, given the agents' utilities for each good. *Fair allocations* in this context are those maximizing the minimum utility received by any agent. We give hardness results and polynomial time approximation algorithms for several variants of this problem. Our main result is a bicriteria approximation in the model with additive utilities, in which a $(1 - \frac{1}{k})$ fraction of the agents receive utility at least OPT/k, for any integer k. This result is obtained from rounding a suitable linear programming relaxation of the problem, and is the best possible result for our LP. We also give an $O(\sqrt{n})$ approximation for a special case with only two classes of goods, an $(m - n + 1)$ approximation for instances with submodular utilities, and extreme inapproximability results for the most general model with monotone utilities.

This research is supported by NSF ITR grants CCR-0122581 and IIS-0121678.

Keywords: Fair Allocation, Indivisibilities, Discrete Allocation, Approximation Algorithms, Generalized Assignment, Lexicographic Matchings

1 Introduction

Economists have long studied many issues surrounding the allocation of goods and services under various economic systems (See e.g. [5, 22, 24, 32, 33]). The two notions of efficiency and fairness of allocations have played a central role in this research area. Despite this, little is known about the computational aspects of finding efficient and/or fair allocations in markets with indivisible goods – that is, goods that cannot be divided and fractionally assigned to multiple agents. Previous work in economics on finding fair allocations has focused largely on the structural properties of markets with indivisible goods, for example, the existence or non-existence of allocations with various properties [17, 4], or the minimum amount of money to offset indivisibilities in the market [1, 9]. Other early work in operations research focused on special cases that are tractable [12, 31], e.g. when there is only one type of good and general utility functions as in [12], or on exponential time algorithms for general models [21]. Yet another strain of research focused on techniques to solve general classes of problems with fairness measures incorporated into the objective function [27, 30]. For example, Sokkalingam and Aneja [30] solved a variety of optimization problems with lexicographic objective functions, including bipartite matching. Such lexicographically maximal matchings yeild a fair allocation when there is exactly one good per agent.

In computer science, work on finding fair allocations has focused on graph based problems and load balancing. For example, Megiddo [23] considered the problem of finding flows in a graph with multiple sources and sinks that fairly distribute the flow absorbed by each sink, and Kleinberg, Rabani, and Tardos [16] consider the problem of finding allocations of bandwidth that ensure fair routing. Fairness in load balancing has been studied for several problems, such as scheduling jobs to unrelated machines to minimize the maximum processing time of any machine [11, 19, 29], and facility location to minimize the maximum distance from a client to its closest open facility [18].

Other work has focused on the complexity of finding allocations with other properties of note. For example, Gale and Shapley [10] have investigated stability of allocations, Irving, Leather, and Gusfield [13] considered finding fair stable marriages, Scarf and Shapley [28] and Pál and Tardos [26] have investigated group strategy proofness, and Jain and Vazirani [15] equitable cost shares. Recently, the question of finding envy-free or envy minimizing allocations of indivisible goods has been investigated by Lipton et al. [20]. On the complexity side, there has been work investigating the existence and computational complexity of finding equilibria in markets [6, 7, 8, 14, 28].

In this paper, we investigate the complexity of finding fair allocations in markets composed entirely of indivisible goods. We call an allocation maximizing the minimum utility received by any agent *max-min fair*, and differentiate it from the notion of *max-min fairness* commonly used in the literature, which requires an allocation to be utility-wise *lexicographically maximum*: that is, an allocation in which the minimum utility received by any agent is maximized, and subject to this, the minimum utility received by the remaining agents is maximized, and so on. We will call this latter notion of fairness *strong max-min fairness.* We investigate this Max-Min Allocation (MMA) problem with several types of utility functions, e.g. monotone and additive functions (see definitions below). Finding approximate max-min fair allocations for agents with additive utilities was considered independently by Bezáková and Dani [3], who give a randomized 2 approximation for instances with two agents, and an $(m-n+1)$ approximation for an instance with m goods and n agents. We improve upon both of these results.

Summary of Results. For MMA with general monotone utilities, n agents, and m goods, we show the task of computing an $f(n,m)$ approximation, for any polynomial time computable function $f(n,m)$, is $\mathbf{NP}-$hard and has communication complexity $\Omega(2^m m^{-\frac{1}{2}})$, even with only two agents. These results force us to retreat to the case of simpler utility functions.

In the additive utility model, where the utility of an agent a for good g is denoted u_{ag}, we give a deterministic algorithm yielding a bicriteria solution that given any $k \in \mathbb{Z}_+$ returns an allocation giving at least $\lceil (1-\frac{1}{k})n \rceil$ agents utility at least $\frac{\mathsf{OPT}}{k}$. Our algorithm is based on rounding a suitable linear programming relaxation of the problem. We also note that this bicriteria result is the best result possible for our LP; in general there does not exist an allocation rounded from the LP giving $\lceil (1-\frac{1+\epsilon}{k})n \rceil$ agents utility at least $\frac{1+\epsilon}{k}\mathsf{OPT}$, for any fixed k.

Using min-cut and feasible flow techniques, we obtain an $O(\sqrt{n})$ approximation for a variant of MMA in which there are "small" goods that each agent values at zero or one, and "big" goods that each agent values at zero or x, where $x > 1$ (this is called Big Goods/Small Goods MMA).

We additionally prove it is $\mathbf{NP}-$hard to $(2-\epsilon)$ approximate MMA in the additive utilities model even with utilities in $\{0, 1, 2\}$. Note there is a 2 approximation in this case, since it is easy to obtain a $u_{\max}/u_{\min}$ approximation where $u_{\max}$ and $u_{\min}$ are the maximum and minimum non-zero utility values, respectively.

We also show that finding *strongly* max-min fair allocations for the matching version of MMA can be used to obtain a $(m-n+1)$ approximation for MMA instances where each agent's utility function is submodular. Strongly max-min fair matchings can be found by an algorithm of Sokkalingam and Aneja [30], however we provide an alternative conceptually simpler algorithm obtained via a reduction to max weight perfect matching. We note in passing that our (very large) weights are essentially as small as possible for such a reduction.

Organization. The hardness results for MMA are presented in Section 2. The n-approximation and bicriteria solution for additive MMA, based on LP rounding, are in Section 3. Section 4 contains the $O(\sqrt{n})$ approximation for Big Goods/Small Goods MMA, based on flow techniques. Section 5 contains the algorithm to find strongly max-min fair matchings, and its extension to a $(m-n+1)$ approximation for submodular MMA. All proofs not appearing in their relevant sections are in the appendices.

Notation and Definitions. In this paper, n indicates the number of agents, and m the number of goods. An allocation is a function π from agents $\mathcal{A}$ to sets of goods, such that $\{\pi(a) \mid a \in \mathcal{A}\}$ is a partition of the goods, $\mathcal{G}$. A utility function $u : \mathcal{A} \times 2^{\mathcal{G}} \to \mathbb{N}$, gives the utility of any agent for any subset of goods. We assume throughout that for any agent a, $u(a, \emptyset) = 0$.

Definition 1. *Given a set of n agents $\mathcal{A}$, a set of m goods $\mathcal{G}$, and a utility function u, the Max-Min Allocation problem (MMA) is to output an allocation π maximizing* $\min_{a\in\mathcal{A}} u(a, \pi(a))$.

Definition 2. *A utility function is monotone if $\forall a \in \mathcal{A}, \forall S', S$ such that $S' \subseteq S \subseteq \mathcal{G}$, $u(a, S') \leq u(a, S)$. A utility function is submodular if $\forall a \in \mathcal{A}, \forall S', S \subseteq \mathcal{G}$, $u(a, S \cup S') + u(a, S \cap S') \leq u(a, S) + u(a, S')$. A utility function is additive if $\forall a \in \mathcal{A}, \forall S \subseteq \mathcal{G}$, $u(a, S) = \sum_{g\in S} u(a, \{g\})$. In this case, we write $u(a, g)$ or u_{ag} as shorthand for $u(a, \{g\})$. A utility function is value derived if it is additive, and there exists a function $v : \mathcal{G} \to \mathbb{N}$ such that $\forall a \in \mathcal{A}, \forall g \in \mathcal{G},\ u(a, g) \in \{0, v(g)\}$.*

2 Hardness of Max-Min Allocation

All relevant proofs not appearing in this section are in Appendix A.

Theorem 1. *Max-Min Allocation with general monotone utilities, is* $\mathbf{NP}-$hard *to $f(n, m)$ approximate for any primitive recursive function $f(n, m)$, even with only two agents.*

Proof. Let $S \subset \mathbb{N}$ be an instance of the Partitioning problem. Let $\sigma : 2^S \to \mathbb{N}$ be defined by $\sigma(X) := \sum_{x\in X} x$. Recall the Partitioning problem is to determine if $\exists X \subset S$ such that $\sigma(X) = \frac{1}{2}\sigma(S)$. Create a Max-Min Allocation instance with two agents, $\mathcal{A} = \{1, 2\}$, goods $\mathcal{G} = S$ and $\forall X \subseteq S$

$$u(1, X) = u(2, X) = \begin{cases} 1 & \text{if } \sigma(X) \geq \frac{1}{2}\sigma(S) \\ 0 & \text{otherwise} \end{cases}$$

Clearly, there exists a 1-allocation iff $\exists X \subset S$ such that $\sigma(X) = \frac{1}{2}\sigma(S)$. Any allocation that isn't a 1-allocation is a 0-allocation, so an $f(n, m)$ approximation algorithm can distinguish between "yes" and "no" instances of Partition. The claim follows. □

Theorem 2. *Max-Min Allocation with general monotone utilities provided by access to an oracle requires $\Omega(2^m m^{-1/2})$ time to $f(n, m)$ approximate for any primitive recursive function $f(n, m)$, even with only two agents.*

Theorem 2 is a corrolary of the main result of [25], and can be proven directly using Nisan and Segal's techniques (see Appendix A for details).

In light of the severe inapproximability of MMA with monotone utilities, we consider only MMA with submodular, additive, or value derived utilities for the remainder of the paper. However, even in the Big Goods/Small Goods model, with the big goods valued at zero or two, MMA remains hard.

Lemma 1. *Max-Min Allocation is* $\mathbf{NP}-$hard *to $(2 - \epsilon)$ approximate for any $\epsilon > 0$, even in the Big Goods/Small Goods model, even if the utilities are derived from values $v : \mathcal{G} \to \{1, 2\}$.*

3 Additive Max-Min Allocation Via LP Rounding

We consider a linear programming relaxation of the associated integer program for MMA with additive utilities. Using a deterministic rounding procedure, we obtain an n-approximation in this fashion, and show that the linear program has integrality gap n. Lastly, we give an efficient algorithm that, given any $k \in \mathbb{Z}_+$, computes an allocation giving $\lceil (1 - \frac{1}{k})n \rceil$ agents utility at least $\frac{\mathsf{OPT}}{k}$, and show that this is tight for the LP considered. All relevant proofs not appearing in this section are in Appendix B.

3.1 The LP, and When It Works Well

The following IP, denoted IP_0, represents additive MMA instances precisely.

$$\begin{array}{ll} \textbf{maximize } \lambda & \\ x_{ag} \in \{0,1\}, & \forall a \in \mathcal{A}, g \in \mathcal{G} \\ \sum_{a \in \mathcal{A}} x_{ag} = 1, & \forall g \in \mathcal{G} \\ \sum_{g \in \mathcal{G}} u_{ag} x_{ag} \geq \lambda, & \forall a \in \mathcal{A} \end{array} \tag{1}$$

A solution $\vec{x}$ allocates good g to agent a iff $x_{ag} = 1$. Let $IP_0^* \equiv \mathsf{OPT}$ denote the maximum λ for which the above IP has a solution. Let LP_0 denote the LP relaxation obtained by replacing $x_{ag} \in \{0,1\}$ with $0 \leq x_{ag} \leq 1$ for all a and g. Let LP_0^* denote the maximum obtainable λ in LP_0. We make some observations about this LP.

We first show that if there is a solution X to LP_0, then there is a solution $\hat{X}$ giving all agents at least as much utility as they received under X, and has the property that the undirected graph $G_{\hat{X}}$ defined below is a forest. $G_{\hat{X}}$ has vertex set $V_{\hat{X}} = \mathcal{A} \cup \mathcal{G}$, and edges $E_{\hat{X}} = \{(a,g) | x_{ag} > 0\}$. Let $\vec{U}(X)$ be the vector of utilities received by agents under a fractional allocation X, sorted in non-decreasing order, and let $\vec{U}_i(X)$ be the i^{th} component of $\vec{U}(X)$.

Lemma 2. *Given feasible LP_0 solution X, there is a polynomial time algorithm to compute a solution $\hat{X}$ such that for each i, $\vec{U}_i(X) \leq \vec{U}_i(\hat{X})$ and $G_{\hat{X}}$ is a forest.*

Note that the integrality gap of LP_0 is infinite. For consider an instance with agents $\mathcal{A} = \{1, 2, \ldots, n\}$ and one good, g, such that every agent values g at n, i.e. $u_{ag} = n$ for all agents a. Then the best integral allocation is a 0-allocation, and the best fractional allocation, given by $x_{ag} = \frac{1}{n}$ for all $a \in \mathcal{A}$, is a 1-allocation. We circumvent this problem by defining $u_{ag}^{\kappa} := \min\{\kappa, u_{ag}\}$ for $\kappa \in \mathbb{N}$, and letting $LP(\kappa)$ be the following LP.

$$\begin{array}{ll} \textbf{maximize } \lambda & \\ 0 \leq x_{ag} \leq 1, & \forall a \in \mathcal{A}, g \in \mathcal{G} \\ \sum_{a \in \mathcal{A}} x_{ag} = 1, & \forall g \in \mathcal{G} \\ \sum_{g \in \mathcal{G}} u_{ag}^{\kappa} x_{ag} \geq \lambda, & \forall a \in \mathcal{A} \end{array} \tag{2}$$

Let $IP(\kappa)$ denote the corresponding integer program with $x_{ag} \in \{0,1\}$. As before, define $LP(\kappa)^*$ and $IP(\kappa)^*$ as the maximum obtainable λ for $LP(\kappa)$ and $IP(\kappa)$, respectively. Binary search for the maximum κ such that $LP(\kappa)^* \geq \kappa$. Let κ^* be this value, and define $LP_1 := LP(\kappa^*)$, $IP_1 := IP(\kappa^*)$. We will work with LP_1 rather than LP_0, and demonstrate that the integrality gap of LP_1 is exactly n. First we must make some observations about LP_1 and IP_1. Note that Lemma 2 applies to LP_1, since it applies to LP_0 for any MMA instance $\mathcal{I}$, and for any LP_1 we can construct an instance $\mathcal{I}'$ such that $LP_0' = LP_1$. Lemma 11 in Appendix B proves that $IP_0^* \leq LP_1^* = \kappa^*$.

We can characterize especially hard instances of additive MMA as those in which some agent must receive virtually all of its utility from a single good, by giving a constant factor approximation algorithm if such is not the case.

Lemma 3. *If for all (a, g) pairs $u_{ag} \leq \beta \cdot LP_0^*$ then there is a polynomial time algorithm, that returns a $(1-\beta)LP^*$ allocation, and hence a $\frac{1}{1-\beta}$ approximation for IP_0.*

We sketch the algorithm as follows: Solve the LP, obtaining an optimal solution X. Convert X to $\hat{X}$ such that $G_{\hat{X}}$ is a forest as described in Lemma 2. For each tree T in $G_{\hat{X}}$, root T at any agent arbitrarily. Allocate each agent its children in T.

3.2 A n-Approximation Algorithm

Here we give an upper bound of n to the integrality gap of LP_1 by giving an n approximation algorithm. Call an agent c-happy if that agent receives utility at least c under the current allocation. Let $\rho(T) := \frac{\kappa^*}{|\mathcal{A} \cap V[T]|}$.

Algorithm: ALLOCATE()

Solve LP_1 optimally, obtaining solution X.
Convert X to $\hat{X}$ as described in Lemma 2.
for each tree T of $G_{\hat{X}}$
  Root T at some agent arbitrarily.
  for each agent a of T from the bottom of the tree up (e.g. in postfix order)
    Allocate an agent a's leaf goods to it.
    if an agent is $\rho(T)$-happy under the partial assignment
      then remove it and its leaf children.
    else $\begin{cases} \text{allocate its parent to it, as well as its leaf children,} \\ \text{and remove it, its parent, and its leaf children.} \end{cases}$

Theorem 3. *The above algorithm yields an n approximation.*

Proof. Since $LP_1^* = \kappa^*$, an optimal solution $\hat{X}$ gives every agent utility at least κ^*. WLOG, assume $G_{\hat{X}}$ is a single tree T, and $n = |\mathcal{A} \cap V[T]|$. Since we will not need to consider X further, let x_{ag} be the fractional amount of g that $\hat{X}$ gives to a. Note that agents cannot be leaves, unless they are allocated all of a good they esteem at κ^* and nothing else. In the latter case, merely allocate them the good the LP did, and nothing else. Otherwise, since the LP gives them utility at least κ^* and $(\forall a, g)\ u_{ag} \leq \kappa^*$, they must get at least some fractional part of multiple goods. Thus they have degree at least two, and are internal nodes in T.

As the algorithm progresses, an agents' internal children may be removed. An internal good is removed in this way only when it is allocated to one of its children. We call such an internal child good *taken* from its parent, and the (fractional) utility to the parent *stolen*. Thus, if good g was taken from agent a by agent b, then the stolen utility amounts to $u_{ag}x_{ag}$. The other two components of an agent's utility in T are its remaining children and its parent. Fix some execution of the algorithm. Let c_a be the utility given to agent a by its non-stolen children in the allocation computed by the algorithm, and let p_a be the utility its parent gives it in T, i.e. if g is the parent of a, $p_a = u_{ag}x_{ag}$. Let s_a denote the utility stolen from a. Thus $s_a + p_a + c_a \geq \kappa^*$, since $\hat{X}$ is an optimal solution to LP_1.

An agent a gets less than $\frac{\kappa^*}{n}$ utility iff $c_a < \frac{\kappa^*}{n}$ and a's parent good g is allocated to one of a's siblings, b. Agent b can only be allocated g if $c_b < c_a < \frac{\kappa^*}{n}$. However, we will show $\max\{c_a, c_b\} \geq \frac{3\kappa^*}{2n}$. Note that a and b must have an agent ancestor since the tree is rooted at an agent, and thus $s_a + s_b \leq \frac{d_a+d_b}{n}\kappa^* \leq (1 - \frac{3}{n})\kappa^*$ by Lemma 4. Further, g can provide at most κ^* utility in total, so $p_a + p_b \leq \kappa^*$, and $s_a + c_a + p_a + s_b + c_b + p_b \geq 2\kappa^*$ since $\hat{X}$ is a κ^* fractional allocation. Thus $c_a + c_b \geq \frac{3\kappa^*}{n}$. So every agent gets utility at least $\frac{\kappa^*}{n}$. $\square$

The following lemma is crucial for the proof of Theorem 3.

Lemma 4. *Fix any tree T of $G_{\hat{X}}$, and root it at an agent. Let $n' = |\mathcal{A} \cap V[T]|$. For each agent $a \in V[T]$, let d_a be the number of agent descendants of a (excluding a) in T. If $\hat{X}$ is a fractional κ^*-allocation, then in any fixed execution of Algorithm* Allocate()*, $s_a \leq \frac{d_a}{n'}\kappa^*$ for all agents a.*

Proof. Consider any tree T in $G_{\hat{X}}$ and let $n' = |\mathcal{A} \cap V[T]|$. Proceed by induction on d_a. For the basis, consider an agent a with $d_a = 0$. It's only children are leaf goods, since $G_{\hat{X}}$ is bipartite. Thus $s_a = 0 \leq d_a/n$ since only non-leaf goods can be stolen from an agent. For the induction step, consider an agent a with internal children goods $\{g_1, \ldots, g_t\}$ taken by agents $\{1, 2, \ldots, t\}$. Agent i could only have taken good $g = g_i$ from a if $c_i < \frac{\kappa^*}{n'}$. By the induction hypothesis, $s_i \leq \frac{d_i}{n'}$, thus $c_i + p_i + s_i \geq \kappa^*$ implies $u_{ig}x_{ig} = p_i > (1 - \frac{d_i+1}{n'})\kappa^*$. This in turn implies $u_{ag}x_{ag} < \frac{d_i+1}{n'}\kappa^*$. Therefore the total utility stolen from a is

$$s_a = \sum_{i=1}^{t} u_{ag_i}x_{ag_i} \ < \ \kappa^* \sum_{i=1}^{t} \frac{d_i+1}{n'} \quad \leq \frac{d_r}{n'}\kappa^*$$

which completes the induction. Note that if $d_i > 0$, then in fact $s_i < \frac{d_i}{n'}\kappa^*$. $\square$

Lemma 5. *The integrality gap of LP_1, the maximum of $\frac{LP_1^*}{IP_1^*}$ over all MMA instances, is exactly n.*

3.3 Allowing for Poverty: Bicriteria Solutions

Here we consider the case when there exist other mechanisms to compensate some fixed fraction of the agents, or when it is acceptable for some of the agents to receive nothing. Call such agents *poor*. Given some number of agents that we may mark as poor, we seek to maximize the minimum utility received by those agents not marked poor. In this, the main result of the paper, we show that the deterministic algorithm below gives $\lceil (1-\frac{1}{k})n \rceil$ agents utility at least $\frac{\mathsf{OPT}}{k}$, for any $k \in \mathbb{Z}_+$, and that our LP precludes any better results via rounding. We annotate the algorithm to simplify the proof. The significance of the tokens is explained below.

Algorithm: BICRITERIA-ALLOCATE(k)

Construct LP_1 and find an optimal acyclic solution $\hat{X}$ for it as in Algorithm ALLOCATE()
for each tree T in $G_{\hat{X}}$
  Root T at some agent arbitrarily.
  for each agent a of T, in postfix order
    Allocate an agent a's leaf goods to it.
    if a is now $\frac{\kappa^*}{k}$-happy
      then remove it and its leaf goods from the graph.
    else if giving agent a its parent good p as well will *not* make it $\frac{\kappa^*}{k}$-happy
      then mark agent a "poor" and remove it and its leaf goods.
    else { **if** p is currently unallocated
      then allocate p to a and remove a and all goods allocated to it.
      else mark agent a as "waiting for siblings."
    if all of a's siblings are either waiting for siblings or have been removed
      then { mark agent a and all its waiting siblings poor, remove them, and add b tokens to p's parent agent a', where $u_{a'g}x_{a'g} \in (\kappa^*\frac{b-1}{k}, \kappa^*\frac{b}{k}]$.

The algorithm finds κ^* and constructs LP_1, solves it and converts the solution into an "acyclic" solution (i.e., one in which G_X is a forest). The algorithm then proceeds up the tree, in e.g. a postorder traversal of the agents, allocating an agent's leaf goods to it, checking if this makes it $\frac{\kappa^*}{k}$-happy, and allocating its parent good to it if not. The algorithm continues in this way until it cannot make an agent $\frac{\kappa^*}{k}$-happy in this way, either because its remaining children and parent are insufficient (denoted event I), or because its parent has already been allocated to some other agent (called event II). In either event, simply mark the agent as poor, remove it and the goods allocated to it, and continue. (We make the distinction between event I and event II only to facilitate the proof.) Ideally, at most $1/k$ fraction of the agents from each subtree would be marked poor. Since this is not the case, an accounting mechanism is needed, in which subtrees *donate* and *receive* agents from other subtrees. The tokens in the algorithm serve this purpose; each token denotes a donated agent. Thus if don(T) agents are donated by a tree T (i.e. added to some agent's account not in T when an agent in T

was removed), $\mathrm{rec}(T)$ agents are received by T from other subtrees, and $\mathrm{poor}(T)$ agents in T are marked poor, the algorithm ensures that $|\mathcal{A} \cap V[T]| + \mathrm{rec}(T) - \mathrm{don}(T) \geq k \cdot \mathrm{poor}(T)$.

Theorem 4. *Given any $k \in \mathbb{Z}_+$, the above algorithm yields an allocation giving at least $\lceil (1 - \frac{1}{k})n \rceil$ agents utility at least $\frac{\mathsf{OPT}}{k}$.*

Given an LP solution X, we say an algorithm *respects* X if its output allocation π gives $g \in \mathcal{G}$ to $a \in \mathcal{A}$ only if $x_{ag} > 0$.

Theorem 5. *Given any $k \in \mathbb{Z}_+$ and any optimal LP solution X, no algorithm respecting X can yield an allocation giving at least $\lceil (1 - \frac{1+\epsilon}{k})n \rceil$ agents utility at least $\frac{1+\epsilon}{k}\mathsf{OPT}$ for any $\epsilon > 0$.*

4 Big Goods/Small Goods Max-Min Allocation.

In this section we consider flow based algorithms. We give an x approximation in the case that $u_{ag} \in \{0\} \cup [1, x]$ for some $x \geq 1$, and an $O(\sqrt{n})$ approximation algorithm for the Big Goods/Small Goods MMA variant, in which goods can be partitioned into two sets $\mathcal{G} = \mathcal{G}_B \uplus \mathcal{G}_S$ such that for some $x > 1$, for all agents $a \in \mathcal{A}$, $g^B \in \mathcal{G}_B$, $g^S \in \mathcal{G}_S$, $u(a, g^S) \in \{0, 1\}$ and $u(a, g^B) \in \{0, x\}$. All relevant proofs not appearing in this section are in Appendix C. We begin with the following observations.

Lemma 6. *Max-Min Allocation instances in which $u_{ag} \in \{0, 1\}$ for all $a \in \mathcal{A}, g \in \mathcal{G}$ can be solved in polynomial time, using max-flow computations.*

Corollary 1. *Max-Min Allocation instances in which there exists $x > 1$ such that $u_{ag} \in \{0\} \cup [1, x]$ for all $a \in \mathcal{A}, g \in \mathcal{G}$ can be x-approximated in polynomial time, by solving the instance exactly using utilities $u'_{ag} = \min\{u_{ag}, 1\}$.*

The following is a generalization of Hall's theorem that will be needed.

Lemma 7. *Given a set $\mathcal{A}$ of n agents, nV goods, and additive utilities $u_{ag} \in \{0, 1\}$, let $S_a := \{g \mid u_{ag} = 1\}$, $S_I := \{g \mid (\exists a \in I)(u_{ag} = 1)\} = \cup_{a \in I} S_a$, and $f(I) := |S_I|$. If $(\forall I \subseteq \mathcal{A},\ I \neq \emptyset)(f(I) \geq V|I|)$ then there exists a V-allocation on these goods.*

Given an instance of Big Goods/Small Goods MMA, with $u_{ag} \in \{0, 1, x\}$ for all a and g, we assume that the optimum satisfies $\mathsf{OPT} \in [x, 2x]$ and that $x \geq \sqrt{n}$. We justify these assumptions later, in the proof of Theorem 6. For now, let b be the number of agents receiving big goods in some x allocation (henceforth called "big agents"), and $s = n - b$ be the number of agents receiving only small goods in that x allocation (henceforth called "small agents").

Lemma 8. *There is a polynomial time algorithm that yields a $\lfloor \frac{x}{s} \rfloor$ allocation.*

Proof. Let $\mathcal{A}_B$ be the set of agents i with $|S_i| < x$. The algorithm is as follows: Find a maximum cardinality matching from big goods to agents containing $\mathcal{A}_B$ in the matching, and allocate the big goods accordingly. Let $\mathcal{A}'_B$ be the set of agents receiving big goods. Allocate the small goods to the remaining agents, $\mathcal{A}_C := \mathcal{A} - \mathcal{A}'_B$, in the fairest way possible, via a feasible flow algorithm.

The agents in $\mathcal{A}'_B$ have utility x. Note that $|\mathcal{A}'_B| \geq b$, so $|\mathcal{A}_C| \leq s$. Since each agent i in $\mathcal{A}_C$ has $|S_i| \geq x$, $(\forall I \subseteq \mathcal{A},\ I \neq \emptyset)(f(I) \geq x)$. We conclude that $(\forall I \subseteq \mathcal{A}_C)(f(I) \geq \frac{x}{|\mathcal{A}_C|}|I|)$. Applying Lemma 7 with $V = \frac{x}{|\mathcal{A}_C|}$, there must be a $\lfloor \frac{x}{|\mathcal{A}_C|} \rfloor$ allocation on the agents of $\mathcal{A}_C$ using only small goods. Since $s \geq |\mathcal{A}_C|$, and the $u_{ag} \in \{0, 1\}$ case is solved exactly, the above algorithm returns a $\lfloor \frac{x}{s} \rfloor$ allocation. □

The $O(\sqrt{n})$ approximation algorithm splits up the instance into two subproblems. In one of these subproblems, a large fraction of the agents (at least $1 - \frac{1}{\sqrt{n}}$) will receive big goods. The other subproblem can be approximated adequately with only small goods. Recall $S_a := \{g \mid u_{ag} = 1\}$, $S_I := \cup_{a \in I} S_a$, and $f(I) := |S_I|$. The following is a useful subroutine to find the maximum cardinality subset $J \subseteq \mathcal{A}$ such that $f(J) \leq c|J|$.

Construct a bipartite graph G on the agents and goods, with $(a, g) \in E[G]$ if $u_{ag} = 1$, having unit capacity. Add a sink node t with $(g, t) \in E$ for each good g, also having unit capacity. Finally add a source node s, with edges (s, a) to all agents a of capacity c. Compute a min-cut, and output the set of agents in the source side of the min-cut.

Algorithm: ROOT-N-APPROX()

---

comment: Let $\mathcal{A}_B = \{i \mid \ (|S_i| < x)\}$. These agents must be big in any x allocation.

Remove $\mathcal{A}_B$ from the instance for now. Let $\mathcal{A}_R$ be the set of agents that remain.
Find the max cardinality subset $J \subseteq \mathcal{A}_R$ such that $f(J) \leq \frac{x|J|}{\sqrt{n}}$ as described above.
comment: Subproblem 1 contains agents in $J \cup \mathcal{A}_B$, all big goods, and goods in S_J.

Solve subproblem 1 using the $\lfloor \frac{x}{s} \rfloor$ approximation algorithm given above.
comment: Subproblem 2 will involve all the other agents and all the leftover small goods.

Solve subproblem two directly using the flow algorithm.

Lemma 9. *The algorithm above yields a $\lfloor \frac{x}{\sqrt{n}} \rfloor$ allocation.*

Proof. Consider subproblem 1. Since $f(J) \leq \frac{x|J|}{\sqrt{n}}$, in the x allocation fewer than $\frac{|J|}{\sqrt{n}}$ agents in J are small. Thus in this instance $s \leq |J|/\sqrt{n} \leq n/\sqrt{n} = \sqrt{n}$, and we can obtain a $\lfloor \frac{x}{\sqrt{n}} \rfloor$ allocation for the agents in this subproblem (i.e. $J \cup \mathcal{A}_B$). Now consider subproblem 2. Let $\overline{J} := \mathcal{A}_R - J$. Note that for all $K \subseteq \overline{J}$, it must be that $|S_K - S_J| > \frac{x|K|}{\sqrt{n}}$. If not, then $f(J \cup K) \leq (x|J|/\sqrt{n}) + (x|K|/\sqrt{n})$. Since $|J| + |K| = |J \cup K|$, this would violate the maximality of J. Applying Lemma 7, there is a $\lfloor \frac{x}{\sqrt{n}} \rfloor$ allocation in this subproblem using only small goods not wanted by any agents in J. Since it uses only small goods, we can solve it directly using the flow algorithm to obtain such an allocation. □

Theorem 6. *There is a polynomial time $O(\sqrt{n})$ approximation algorithm for Big Goods/Small Goods Max-Min Allocation.*

Proof. Assuming $\mathsf{OPT} \in [x, 2x]$ and $x \geq \sqrt{n}$, Root-N-Approx returns a $\lfloor \frac{x}{\sqrt{n}} \rfloor$ allocation, and so $x \geq \sqrt{n}$ and $\mathsf{OPT} \leq 2x$ imply $\lfloor \frac{x}{\sqrt{n}} \rfloor > \frac{x}{2\sqrt{n}} \geq \frac{\mathsf{OPT}}{4\sqrt{n}}$. So under these assumptions, we obtain a $4\sqrt{n}$ approximation. We can dismiss these assumptions as follows: If $\mathsf{OPT} \geq 2x$, then by Lemma 3, we can obtain a 2 approximation with a LP based algorithm. If $x < \sqrt{n}$, we can obtain a $\sqrt{n}$ approximation via Corollary 1. Finally, if $\mathsf{OPT} < x$, we can run the algorithm again using $u'_{ag} = \frac{x}{2}$ if $u_{ag} = x$, and $u'_{ag} = u_{ag}$ otherwise. $\square$

5 Finding Strongly Max-Min Fair Matchings.

We conclude our algorithmic investigations of MMA with a polynomial time algorithm to find *strongly* max-min fair allocations when $m = n$.

A strongly max-min fair matching is defined as follows. For any allocation π, let $U(\pi) = \{u(a, \pi(a)) \mid a \in \mathcal{A}\}$ be the multiset of utilities received by the agents. Let $\vec{U}(\pi) \in \mathbb{N}^n$ be the vector of utilities in $U(\pi)$, sorted in non-decreasing order. Let $\vec{U}_i(\pi)$ denote the i^{th} coordinate of $\vec{U}(\pi)$. We write π is *fairer* than π', denoted $\pi \succ \pi'$, if $\exists j$ such that $\vec{U}_j(\pi) > \vec{U}_j(\pi')$ and $\forall i < j$, $\vec{U}_i(\pi) = \vec{U}_i(\pi')$. We write π is *as fair as* π', denoted $\pi \succeq \pi'$ if $\pi \succ \pi'$ or $\vec{U}(\pi) = \vec{U}(\pi')$. An allocation π is *strongly max-min fair* if $\forall \pi', \pi \succeq \pi'$.

Note that for the special case that $m = n$, a *strongly* max-min fair allocation corresponds exactly to a lexicographically maximal matching in a suitably weighted bipartite graph. Such a matching can be found in polynomial time via an algorithm of Sokkalingam and Aneja [30]. We provide an alternative, conceptually simpler algorithm based on reducing lexicographically maximal matchings to maximum weight matchings. A similiar technique is used by Irving et al. [13] to find "lexicographically maximum stable marriages."

The Algorithm: Given a bipartite graph $(\mathcal{A}, \mathcal{G}, E)$, with edge (a, g) weighted by u_{ag}, delete all edges (a, g) such that g is not among agent a's n favorite (highest utility) goods. Let E' be the remaining edges. Let $V := \{u_{ag} | (a, g) \in E'\}$ be the set of utility values on the remaining edges. If u_{ag} is the i^{th} smallest element of V, let $\phi(u_{ag}) := i$. Define $\tau : \mathbb{N} \to \mathbb{N}$ by $\tau(k) := \sum_{i=1}^{k} n^{(n^2 - i)}$, and compute a maximum weight matching on the graph $(\mathcal{A}, \mathcal{G}, E')$ with edge weights $w_{ag} := \tau(\phi(u_{ag}))$.

Theorem 7. *The above algorithm returns a strongly max-min fair allocation when $m = n$.*

Proof. In this proof, we equate matchings and allocations, since any allocation that isn't a matching is a 0-allocation. For a matching M, define its weight to be $w(M) = \sum_{(a,g) \in M} w_{ag}$. It suffices to prove that for any two matchings M and M', $M \succ M'$ implies $w(M) > w(M')$. Suppose that $M \succ M'$. We will show $w(M) - w(M') > 0$.

Let $\vec{U}(M') = \vec{x}$, $\vec{U}(M) = \vec{y}$. We extend the definition of $\succ$ to relate pairs of vectors in $\mathbb{R}^n$ whose values are sorted in non-decreasing order in a natural way: $\vec{a} \succ \vec{b}$ iff ($\exists j$, $\forall i <$

j) ($a_i = b_i$ and $a_j > b_j$), and $\vec{a} \succeq \vec{b}$ if $\vec{a} \succ \vec{b}$ or $\vec{a} = \vec{b}$. Given a fixed MMA instance determining ϕ, let $V = \{u_{ag} | a \in \mathcal{A}, g \in \mathcal{G}\}$ as before, and define $f(a) = \tau(\phi(a))$ for $a \in V$ and $f(\vec{a}) := \sum_{i=1}^{n} \tau(\phi(a_i))$ for $\vec{a} \in V^n$. We give vectors $\vec{r}, \vec{s}$ such that $\vec{x} \preceq \vec{r} \prec \vec{s} \preceq \vec{y}$ and show $w(M') = f(\vec{x}) \leq f(\vec{r}) < f(\vec{s}) \leq f(\vec{y}) = w(M)$.

Suppose that for all $i < k$, $x_i = y_i$ and $x_k < y_k$. Let $u_{\max} = \max_{a,g}\{u_{ag}\}$. Let $\vec{r} = (x_1, x_2, \ldots, x_{k-1}, x_k,\ u_{\max}, u_{\max}, \ldots, u_{\max})$, let $\vec{s} = (y_1, y_2, \ldots, y_{k-1}, y_k, y_k, y_k, \ldots, y_k)$. Then, since f is strictly increasing, it is clear that $f(\vec{x}) \leq f(\vec{r})$ and $f(\vec{s}) \leq f(\vec{y})$. So it suffices to prove $f(\vec{r}) < f(\vec{s})$, which can be done with a strait-forward calculation. □

A $(m-n+1)$ Approximation for Submodular Max-Min Allocation. Merely interpreting an instance of MMA with submodular utilities u as one with additive utilites $u'_{ag} = u(a, \{g\})$ and obtaining a max-min fair matching yields an $(m-n+1)$ approximation to the original instance, even though the matching leaves $m-n$ goods unallocated. We defer the proof of Lemma 10 to Appendix C.

Lemma 10. *Given a submodular instance $\mathcal{I}$ of MMA, a max-min fair matching M on utilities $u'_{ag} = u(a, \{g\})$ is a $(m-n+1)$ approximation to $\mathcal{I}$.*

Near Optimality of Edge Weights. Our algorithm uses nearly optimally sized weights. Consider any function $f : \mathbb{N} \to \mathbb{N}$, obeying the following property: if the weight of a matching, M, is defined as $w(M) = \sum_{(a,g) \in M} f(u_{ag})$, then every maximum weight matching is a strongly max-min fair allocation. The proof of the following theorem appears in Appendix C.

Theorem 8. *For any function f satisfying the above property, there exists an x such that $f(x) \geq n^{n^2-2n+1}$.*

6 Conclusions

We have given improved approximation algorithms and hardness results for several variants of the problem of finding max-min fair allocations of indivisible goods. However, there remain many interesting open problems pertaining to the allocation of indivisible goods. For example, there is still a large gap in the known approximation hardness of the additive and value derived variants of the Max-Min Allocation problem. The highly constrained Big Goods/Small Goods variant in particular may yield insight into at least one core source of hardness in the problem. Lastly, it remains open whether there exists a way to strengthen the LP for additive Max-Min Allocation with polynomial time separable constraints, to overcome the deficiencies inherent in the basic LP formulation.

Acknowledgments. I thank Jon Kleinberg, Anupam Gupta, and Éva Tardos for helpful discussions and thoughtful guidance, and Kedar Dhamdhere for helpful suggestions. I additionally thank Jon Kleinberg for a simplified proof of Lemma 7 and Éva Tardos for suggesting the cycle-canceling proof of Lemma 2.

References

[1] Ahmet Alkan, Gabrielle Demange, and David Gale. Fair allocation of indivisible goods and criteria of justice. *Econometrica*, 59(4):1023–1039, 1991.

[2] Piotr Berman, Marek Karpinski, and Alexander D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. In *10th Electronic Colloquium on Computational Complexity, Report TR03-049*, 2003.

[3] Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *ACM SIGecom Exchanges*, 5(3):11–18, 2005.

[4] Steven Brams, Paul Edelman, and Peter Fishburn. Fair division of indivisible items. *Theory and Decision*, 55(2):147–180, 2003.

[5] Steven Brams and Alan Taylor. *Fair Division.* Cambridge University Press, New York, NY, 1996.

[6] Vincent Conitzer and Tuomas Sandholm. Complexity results about Nash equilibria. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 765–771, 2003.

[7] Xiaotie Deng, Christos Papadimitriou, and Shmuel Safra. On the complexity of equilibria. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing*, pages 67–71. ACM Press, 2002.

[8] Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 389–395. IEEE Computer Society, 2002.

[9] Satoru Fujishige and Zaifu Yang. Existence of an equilibrium in a general competitive exchange economy with indivisible goods and money. *Annals of Economics and Finance*, 3:135–147, 2002.

[10] David Gale and Lloyd Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[11] Ashish Goel, Adam Meyerson, and Serge Plotkin. Approximate majorization and fair online load balancing. In *Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–390. Society for Industrial and Applied Mathematics, 2001.

[12] Toshihide Ibaraki and Naoki Katoh. *Resource Allocation Problems: Algorithmic Approaches.* MIT Press, Cambridge, MA, 1998.

[13] Robert W. Irving, Paul Leather, and Dan Gusfield. An efficient algorithm for the "optimal" stable marriage. *J. ACM*, 34(3):532–543, 1987.

[14] Kamal Jain, Mohammad Mahdian, and Amin Saberi. Approximating market equilibria. In *Proceedings of the sixth International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2003.

[15] Kamal Jain and Vijay V. Vazirani. Equitable cost allocations via primal-dual-type algorithms. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing*, pages 313–321. ACM Press, 2002.

[16] Jon Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 568. IEEE Computer Society, 1999.

[17] Hideo Konishi, Thomas Quint, and Jun Wako. On the Shapley-Scarf economy: The case of multiple types of indivisible goods. *Journal of Mathematical Economics*, 35:1–15, 2001.

[18] Amit Kumar and Jon Kleinberg. Fairness measures for resource allocation. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 75–85. IEEE Computer Society, 2000.

[19] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[20] Richard Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *The ACM Conference on Electronic Commerce, to appear*, 2004.

[21] Hanan Luss. On equitable resource allocation problems: A lexicographic minimax approach. *Operations Research*, 47(3):361–378, 1999.

[22] Iain McLean and Arnold Urken. *Classics of Social Choice*. U. Michigan Press, Ann Arbor, MI, 1995.

[23] Nimrod Megiddo. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7(3):97–107, 1974.

[24] Hervé Moulin. *Cooperative Microeconomics*. Princeton University Press, Princeton, NJ, 1995.

[25] Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting lindahl prices. In *DIMACS Workshop on Computational Issues in Game Theory and Mechanism Design*, 2001.

[26] Martin Pál and Éva Tardos. Group strategyproof mechanisms via primal-dual algorithms. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 584–593. IEEE Computer Society, 2003.

[27] Marc Posner and Chu-Tao Wu. Linear max-min programming. *Mathematical Programming*, 20:166–172, 1981.

[28] Herbert Scarf and Lloyd Shapley. On cores and indivisibility. *Journal of Mathematical Economics*, 1(1), 1974.

[29] David B. Shmoys and Éva Tardos. Scheduling unrelated machines with costs. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–454. Society for Industrial and Applied Mathematics, 1993.

[30] P. T. Sokkalingam and Y. P. Aneja. Lexicographic bottleneck combinatorial problems. *Operations Research Letters*, 23(1-2):27–33, 1998.

[31] Chistopher S. Tang. A max-min allocation problem: Its solutions and applications. *Operations Research*, 36(2):359–367, 1988.

[32] Barry R. Weingast and Donald Wittman, editors. *Oxford Handbook of Political Economy*, chapter Fair Division. Oxford University Press, 2005.

[33] H. Peyton Young. *Equity*. Princeton University Press, Princeton, NJ, 1994.

A Hardness Proofs

Proof of Theorem 2. This proof is inspired by work in [25]. Let $\mathcal{G}$ be a set of goods with $|\mathcal{G}| = 2k$. Let $\bar{S} = \mathcal{G} - S$ for $S \subseteq \mathcal{G}$. Consider utility functions of type $u : \mathcal{A} \times 2^{\mathcal{G}} \to \{0, 1\}$ obeying the following property for every $a \in \mathcal{A}$ and $S \subseteq \mathcal{G}$

$$u(a, S) = \begin{cases} 0 & \text{if } |S| < k \\ 1 & \text{if } |S| > k \\ (u(a, \bar{S}) + 1) \bmod (2) & \text{if } |S| = k \end{cases}$$

Consider the utility function for a fixed agent, i.e. let $u_a(S) := u(a, S)$. A simple argument given in [25] shows there to be exactly $2^{\binom{2k}{k}}$ such functions u_a. Given any such function $u_a : 2^{\mathcal{G}} \to \{0, 1\}$ obeying the above property and any sized k set of goods, S^*, we can construct a function $u_b : 2^{\mathcal{G}} \to \{0, 1\}$ such that $\forall S \subseteq \mathcal{G}, S \neq S^*$, $u_b(S) = u_a(S)$ and $u_a(S^*) = (u_b(S^*) + 1) \mod (2)$. Consider the instance of MMA with $\mathcal{A} = \{a, b\}$ and utilities $u(a, S) = u_a(S)$, $u(b, S) = u_b(S)$. WLOG, assume $u(a, S^*) = 1$. Then allocating S^* to a and $\bar{S}^*$ to b is a 1-allocation, whereas everything else is a 0-allocation. To obtain a 1-allocation, any algorithm must thus search for a S^* among $\binom{2k}{k}$ possible size k subsets of $\mathcal{G}$. This must take $\Omega(\binom{2k}{k})$ oracle accesses, and thus $\Omega(\binom{2k}{k})$ time. Since $\binom{2k}{k} \approx \frac{1}{\sqrt{\pi k}} 2^{2k}$ via Stirling's approximation, any algorithm must take $\Omega(2^{|\mathcal{G}|}(|\mathcal{G}|^{-1/2}))$ time. Since any $f(n, m)$ approximation algorithm must compute a 1-allocation, the claim follows. □

Proof of Lemma 1. We reduce (3,B2)-SAT to MMA. (3,B2)-SAT is the set of 3-SAT instances in which every literal occurs exactly twice. That it is **NP**−hard follows immediately from [2] in which MAX-(3,B2)-SAT is shown to be **NP**−hard even to approximate beyond some constant. Let F be a (3,B2)-SAT instance with variables $\{x_1, \ldots, x_n\}$ and clauses $\{C_1, \ldots, C_k\}$. Create $2n$ agents $\mathcal{A} = \{(x_i, v) | 1 \leq i \leq n \wedge v \in \{\mathsf{true}, \mathsf{false}\}\}$. Let $\bar{v}$ indicate "not v." Create three groups of goods: n goods with value 2, $\mathcal{G}_B = \{g_1^B, \ldots, g_n^B\}$, k "clause" goods with value 1, $\mathcal{G}_C = \{g_1^C, \ldots, g_k^C\}$, and $2n - k = \frac{2n}{3}$ "dummy" goods with value 1, $\mathcal{G}_D = \{g_1^D, \ldots, g_{2n-k}^D\}$. The utilities are as follows: $u(a, g_i^B) = 2$ if $a \in \{(x_i, \mathsf{true}), (x_i, \mathsf{false})\}$, $u(a, g_i^C) = 1$ if $a = (x_j, v)$ and the boolean assignment of x_j to v causes C_i to evaluate to true, and $u(a, g) = 1$ for all $a \in \mathcal{A}$, $g \in \mathcal{G}_D$. All other utilities $u(a, g)$ are zero. Now note that there is a 2-allocation iff F has a satisfying assignment A. For suppose π is a 2-allocation. Since there are n "big" goods $\mathcal{G}_B$, and $2n$ "small" goods $\mathcal{G}_C \cup \mathcal{G}_D$, a conservation of utility argument implies π gives each agent utility exactly two. Thus every goods $g \in \mathcal{G}_C$ is allocated to an agent not receiving a big good. It follows that $A = \{(x_i, v) | g_i^B \in \pi((x_i, \bar{v}))\}$ is an assignment satisfying F, since every clause is satisfied by some agent receiving only small goods, and one of $(x_i, 0)$ or $(x_i, 1)$ must receive a big good. Conversely, any allocation A gives a 2-allocation π: allocate g_i^B to (x_i, v) if $(x_i, \bar{v}) \in A$. Let $\mathcal{A}_B$ be the agents receiving some big good g_i^B. Allocate goods in G_C to any agent not in $\mathcal{A}_B$ obtaining utility one from them, and goods in $\mathcal{G}_D$ to the remaining agents with utility less than 2. Let $\mathcal{A}_C$ be those receiving utility at least 2 from goods in $\mathcal{G}_C$. Agents in $\mathcal{A}_B$ clearly have utility 2. Further, $\mathcal{A}_B \cap \mathcal{A}_C = \emptyset$, and since each literal appears exactly twice in F, no agent in $\mathcal{A}_C$ is allocated more than 2 goods from $\mathcal{G}_C$. Let $\mathcal{A}_D = \mathcal{A} - \mathcal{A}_B \cup \mathcal{A}_C$. Collectively, $\mathcal{A}_D$ get $k - 2|\mathcal{A}_C|$ goods in $\mathcal{G}_C$. Thus $2|\mathcal{A}_D| - (k - 2|\mathcal{A}_C|) = 2(|\mathcal{A}_C| + |\mathcal{A}_D|) - k = 2n - k$ goods valued at one by everyone suffices to "fill in the gaps" and create a 2 allocation. $\mathcal{G}_D$ does the trick. Furthermore, if there is no 2-allocation, the optimal is then at best a 1-allocation. Thus, for any $\epsilon > 0$, a $(2 - \epsilon)$ approximation can distinguish satisfiable instances of (3,B2)-SAT from unsatisfiable ones, completing the proof. □

B LP Related Proofs

B.1 Properties of LP_0 and LP_1

Proof of Lemma 2. Suppose there is a cycle, C, in G_X. We remove it without decreasing the utility received by any agent by pushing flow around C in order to eliminate an edge e as follows: Suppose $C = a_1g_1a_2g_2 \dots a_kg_ka_1$, where $a_1, \dots, a_k$ are the agents of the cycle, and $g_1, \dots, g_k$ are the goods. Suppose a_1 takes a_k's share of g_k, i.e. set $x_{1k} \leftarrow x_{1k} + x_{kk}$. We consider this as a flow, and set $f_0 = x_{kk}$. To keep a_1's utility fixed, a_1 gives up $u_{1k}x_{kk}$ utility in the form of g_1 to a_2. That is, set $x_{11} \leftarrow x_{11} - f_1$ and $x_{21} \leftarrow x_{21} + f_1$, where $f_1 = \frac{u_{1k}}{u_{11}}x_{kk}$. To keep a_2's utility fixed, a_2 gives up $u_{21}f_1$ utility in the form of g_2 to a_3, and so on. In general, $f_i = \frac{u_{i,i-1}}{u_{ii}}f_{i-1}$ for $i > 0$. Agent a_k thus receives $u_{k,k-1}f_{k-1}$ utility from its additional portion of g_{k-1} in exchange for giving its fraction of g_1 to a_1. Thus agent a_k gains utility $u_{k,k-1}f_{k-1} - u_{kk}f_0$. If this quantity is zero, we have a new allocation achieving the same utilities for each agent, such that $x_{kk} = 0$ and so an edge of C has been removed without adding any edges. If the quantity is positive, a_k is better off than before, and every other agent has the same utility as before. If the quantity is negative, we can merely reverse the direction of flow. That is, we give f_0 units of g_k from a_1 to a_k, f_1 units of g_1 from a_2 to a_1, and so on. If any agent a_i does not have f_{i-1} units of g_{i-1} to give, we can simply multiplicatively scale down all flows by $\frac{x_{i,i-1}}{f_{i-1}}$. This is acceptable since for each i and j, $f_i = c_{ij}f_j$ for some constant c_{ij}. Now the flow is routable, at least one edge is deleted, the utility of a_k increases, and the utility of the other agents remains the same. Repeating this process at most $|E[G_X]|$ times, we find a solution $\hat{X}$ with $G_{\hat{X}}$ acyclic and thus a forest. □

Lemma 11. *$LP_1^* = \kappa^*$ and $IP_0^* \leq LP_1^*$.*

Proof. By definition, $LP_1^* = LP(\kappa^*)^* \geq \kappa^*$. Note that $LP(\alpha)^* \leq LP(\beta)^*$ if $\alpha \leq \beta$. Thus if $LP(\kappa^*)^* = \kappa^* + \epsilon$ for some $\epsilon > 0$, then $LP(\kappa^* + \epsilon)^* = \kappa^* + \epsilon$, contradicting the maximality of κ^*. So $LP_1^* = \kappa^*$. Suppose $IP_0^* > LP_1^*$, so that $IP_0^* = \kappa^* + \epsilon$ for some $\epsilon > 0$. Restricting the utility any good gives to any agent to be under IP_0^* cannot change the value of the optimum, and thus $LP(\kappa^* + \epsilon)^* \geq IP(\kappa^* + \epsilon)^* = \kappa^* + \epsilon$. This contradicts the maximality of κ^*, and so $IP_0^* \leq LP_1^*$. □

Proof of Lemma 3. Let $LP \in \{LP_0, LP_1\}$. Solution $\hat{X}$ is optimal, so each agent a receives $\sum_g u_{ag}x_{ag} \geq LP^*$ utility. Let p be the parent of a in $G_{\hat{X}}$, if it exists. The allocation given by the algorithm gives agent a utility at least $LP^* - u_{ap}x_{ap} \geq LP^* - u_{ap} \geq LP^* - \beta \cdot LP^* = (1-\beta)LP^*$. Thus any agent receives $(1-\beta)LP^*$ utility. Additionally, $IP_0^* \leq LP_0^*$ trivially, and $IP_0^* \leq LP_1^*$ by Lemma 11, and so $\frac{IP_0^*}{(1-\beta)LP^*} \leq \frac{1}{1-\beta}$, proving the second part of the claim. □

Proof of Lemma 5. Since Algorithm Allocate() rounds the LP solution to obtain an n approximation, the integrality gap is at most n. We now show that the integrality gap is at least n. Consider the following value derived MMA instance: $\mathcal{A} = \{1, 2, \dots, n\}$, $\mathcal{G} = \mathcal{G}_S \cup \mathcal{G}_B$ where $\mathcal{G}_S = \{g_1^S, \dots, g_n^S\}$ are small goods with value $v(g) = 1$, and $\mathcal{G}_B = \{g_1^B, \dots, g_{n-1}^B\}$ are big goods with value $v(g) = n$. Set $u_{ag} = n$ for all $a \in \mathcal{A}$, $g \in \mathcal{G}_B$, and $u_{ag} = 1$ if $g = g_i^S$ and $a = i$, and $u_{ag} = 0$ otherwise. Clearly, $IP^* = 1$, since some agent will receive only small goods, and each agent only values one small good at positive utility. However, $LP_1^* \geq n$, since we can allocate g_i^S to agent i for all i, and for each big good g_i^B, we can allocate $\frac{1}{n}$ of it to agent n, and $\frac{n-1}{n}$ of it to agent i. Figure 1 shows the associated LP solution as a graph with edge (a, g) weighted by $u_{ag}x_{ag}$. □

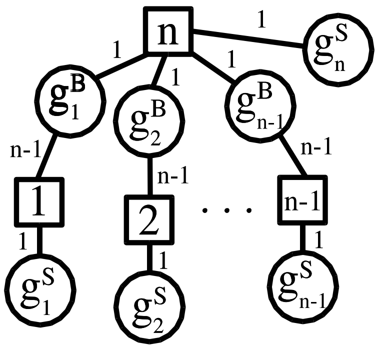

Figure 1: An LP solution with value n. Edges (a, g) are weighted by $u_{ag}x_{ag}$. Agents are squares and goods are circles

Proof of Theorem 5. Consider the tree in figure 1. Let T_n be that tree with n agent nodes. Given any $k \in \mathbb{Z}_+$, consider an MMA instance with $n = ck$ for some $c \in \mathbb{Z}_+$, with LP solution X such that G_X consists of c copies of T_k. Clearly, $LP^* = k$. Consider any tree T_k. If the root agent values all small goods, instead of just one in the case of the proof of Lemma 5, all the small goods can be given to the root, and the big good g_i^B can be given to agent i. This yields an integral k allocation. However, any allocation given by an algorithm respecting X obtains minimum utility at most one. Thus, for such an algorithm to give all non-poor agents utility at least $\frac{1+\epsilon}{k}\mathsf{OPT} = 1 + \epsilon$, at least one agent from each copy of T_k must be poor. Thus at least $\frac{n}{k}$ agents must be poor. But if $\epsilon > 0$ then $\lceil(1 - \frac{1+\epsilon}{k})n\rceil = (n - c + 1)$ and at most $n - c$ agents are non-poor. So it cannot be that $\lceil(1 - \frac{1+\epsilon}{k})n\rceil$ agents get utility at least $\frac{1+\epsilon}{k}\mathsf{OPT}$. $\square$

B.2 Bicriteria Related Proofs

Proof of Theorem 4. By Lemma 11, $\mathsf{OPT} \equiv IP_0^* \leq \kappa^*$, so it suffices to prove that $\lceil(1 - \frac{1}{k})n\rceil$ agents receive utility at least $\frac{\kappa^*}{k}$. Suppose the algorithm above runs Algorithm ALLOCATE() and neither of events I or II occur. Then the algorithm makes every agent $\frac{\kappa^*}{k}$-happy, and we are done.

If event I or event II occur, the algorithm will by then have removed some portion of T from the instance, declared some agents poor, compensated others, and in the case of event II, stolen utility from portions of the tree not yet removed. Fix some execution ξ of the algorithm, and suppose that between initialization and the first event in $\{I, II\}$ the algorithm removes a subtree of T called T_1, and between future consecutive events it removes subtrees $T_2, \ldots, T_q$ labelled in the order they are removed. Thus $T = \bigcup_i T_i$. Let $\text{poor}(T_i)$ be the number of agents in $V[T_i]$ marked poor during execution ξ, and let $\mathcal{A}(T_i) := \mathcal{A} \cap V[T_i]$. We desire that $\frac{\text{poor}(T_i)}{|\mathcal{A}(T_i)|} \leq \frac{1}{k}$ for each T_i. Since this is not always the case, an accounting mechanism is required, so that subtrees with $k \cdot \text{poor}(T_i) = |\mathcal{A}(T_i)| - b$ can "donate" up to b agents to subtrees T_j with $k \cdot \text{poor}(T_j) > |\mathcal{A}(T_j)|$. The tokens keep track of such donations in the following way: if the algorithm gives b tokens to agent $a \in T_j$ upon removing the final elements of T_i, then T_i is said to donate b agents to T_j. We show that for any i, if T_i donated x agents to another subtree under ξ and received y agents as donations from other subtrees under ξ, then $\frac{\text{poor}(T_i)}{|\mathcal{A}(T_i)|+y-x} \leq \frac{1}{k}$. This is sufficient to prove $\frac{\text{poor}(T)}{|\mathcal{A}(T)|} \leq \frac{1}{k}$. Proving all agents not marked poor are $\frac{\kappa^*}{k}$-happy completes the proof.

For agent $a \in \mathcal{A}(T_i)$ we define c_a, s_a and p_a as before but on the tree $T_a := \bigcup_{j \geq i} T_j$. For example, s_a is the utility stolen from a *by agents in* T_a. The quantities c_a and p_a remain the same as before. Define $\text{rec}(T_i) := \sum_{a \in \mathcal{A}(T_i)} \text{rec}_a$, the number of agents donated to T_i, and define $\text{don}(T_i)$ to be number of agents T_i donates to other subtrees. The algorithm maintains two invariants:

1. When T_i is removed, $\frac{\text{poor}(T_i)}{|\mathcal{A}(T_i)|+\text{rec}(T_i)-\text{don}(T_i)} \leq \frac{1}{k}$.

2. For each agent a, $(c_a + p_a + s_a + \frac{\kappa^*}{k}\,\text{rec}_a) \geq \kappa^*$.

The first invariant ensures that with the corrections made by the accounting mechanism, each subtree removed marks at most $1/k$ of its agents poor. The second invariant says that the algorithm compensates agents for "inter-subtree theft" sufficiently so that they still have κ^* utility if each token is worth $\frac{1}{k}$ utility.

We proceed by induction up from the leaves of the tree. For the base case, there is nothing to prove for invariant one, and since $\text{rec}(T_1) = 0$ and $\hat{X}$ is a fractional κ^*-allocation, invariant two holds.

For the induction step, we require a generalization of Lemma 4. For the purpose of bounding s_a for $a \in \mathcal{A}(T_i)$, we can essentially treat tokens in $\text{rec}_{a'}$ as descendants of a'. Define $\texttt{Desc}(a', T)$ as the descendants of agent a', not including a', in tree T. Then by Lemma 12,

$$s_a \leq \frac{\kappa^*}{k} \sum_{a' \in \texttt{Desc}(a,T_i)} (1 + \text{rec}_{a'})$$

Suppose the algorithm removes the remains of T_i upon the occurrence of event I. Note $(c_a + p_a + s_a + \frac{\kappa^*}{k}\,\text{rec}_a) \geq \kappa^*$, which implies $(s_a + \frac{\kappa^*}{k}\,\text{rec}_a) > \kappa^*(1 - \frac{1}{k})$. Applying Lemma 12 yields

$$\begin{aligned} \kappa^*\left(1 - \tfrac{1}{k}\right) &< (s_a + \tfrac{\kappa^*}{k}\,\text{rec}_a) \\ &\leq \tfrac{\kappa^*}{k} \textstyle\sum_{a' \in \texttt{Desc}(a,T_i)} (1 + \text{rec}_{a'}) + \tfrac{\kappa^*}{k}\,\text{rec}_a \\ &= \tfrac{\kappa^*}{k}\,(\text{rec}(T_i) + |\mathcal{A}(T_i)| - 1) \end{aligned}$$

This implies $\text{rec}(T_i) + |\mathcal{A}(T_i)| > k$. Since $\text{poor}(T_i) = 1$ and $\text{don}(T_i) = 0$, invariant one is maintained. Since no utility is stolen from other subtrees, invariant two is maintained.

Lastly, suppose the algorithm removes the remains of T_i upon the occurrence of event II. Some good g has $r \geq 2$ children $1, \ldots, r$ such that $c_i < \frac{\kappa^*}{k}$ for all $i \in \{1, 2, \ldots, r\}$. Let a be g's parent in T, and suppose $u_{ag}x_{ag} \in (\kappa^* \frac{b-1}{k}, \kappa^* \frac{b}{k}]$. Let $S = \{1, 2, \ldots, r\}$, and $N = |\mathcal{A}(T_i)| + \text{rec}(T_i)$. Using invariant two, and summing over all agents in S, we obtain $r\kappa^* \leq \sum_{a \in S}(c_a + p_a + s_a + \frac{\kappa^*}{k}\,\text{rec}_a)$. Combining $c_i < \frac{\kappa^*}{k}$ for all i with $\sum_{a \in S} p_a \leq \kappa^* - u_{ag}x_{ag} < \kappa^*(1 - \frac{b-1}{k})$ yields $r\kappa^* < \frac{r\kappa^*}{k} + \kappa^*(1 - \frac{b-1}{k}) + \sum_{a \in S}(s_a + \text{rec}_a)$. With the bound on s_a given by Lemma 12, we conclude

$$\begin{aligned} r\kappa^* &< \tfrac{r\kappa^*}{k} + \kappa^*(1 - \tfrac{b-1}{k}) + \textstyle\sum_{a \in S}(s_a + \text{rec}_a) \\ &\leq \tfrac{r\kappa^*}{k} + \kappa^*(1 - \tfrac{b-1}{k}) + \textstyle\sum_{a \in S} \text{rec}_a + \tfrac{\kappa^*}{k} \sum_{a \in S} \sum_{a' \in \texttt{Desc}(a,T_i)} (1 + \text{rec}_{a'}) \\ &= \kappa^*(1 - \tfrac{b-1}{k}) + \tfrac{r\kappa^*}{k} + \tfrac{\kappa^*}{k}(|\mathcal{A}(T_i)| - r) + \tfrac{\kappa^*}{k}\,\text{rec}(T_i) \\ &= \kappa^*(1 - \tfrac{b-1}{k}) + \tfrac{\kappa^*}{k}\,(|\mathcal{A}(T_i)| + \text{rec}(T_i)) \\ &= \kappa^*(1 - \tfrac{b-1}{k}) + N\tfrac{\kappa^*}{k} \end{aligned}$$

Thus $rk < N + (k - b + 1)$ and so $N > (r-1)k + b - 1$. Since N is an integer, $N \geq (r-1)k + b$. The first invariant is maintained because $\mathrm{don}(T_i) = b$ and $\mathrm{poor}(T_i) = r - 1$. The second invariant is maintained as well, because $u_{ag}x_{ag} \leq \kappa^* \frac{b}{k}$. We conclude that at most $\frac{1}{k}$ fraction of the agents are marked poor. Finally, the algorithm explicitly guarantees that every agent not marked poor is $\frac{\kappa^*}{k}$-happy, and thus the algorithm gives $\lceil (1 - \frac{1}{k})n \rceil$ agents utility at least $\frac{\kappa^*}{k}$. □

Lemma 12. *For c_a, p_a, s_a and $\mathtt{Desc}(a, T)$ defined in the proof of theorem 4, under any fixed execution of algorithm* Bicriteria-Allocate(), *for every agent a in subtree T_i removed between consecutive events,*

$$s_a \leq \frac{\kappa^*}{k} \sum_{a' \in \mathtt{Desc}(a, T_i)} (1 + \mathrm{rec}_{a'})$$

Proof. Proceed by induction. Let $\Upsilon_a := \sum_{a' \in \mathtt{Desc}(a,T_i)} (1 + \mathrm{rec}_{a'})$. Lemma 4 takes care of the base case in which agent a has $\mathrm{rec}_a = 0$ and $\mathrm{rec}_{a'} = 0$ for all descendants a' of a. For the induction step, Consider an agent $a \in T_i$, where T_i is one of the trees removed by the algorithm between events. Suppose agent a has children goods $\{1, 2, \ldots, r\}$ taken by agents $\{1, 2, \ldots, r\}$ in T_i. Agent j could only have taken good j from a if $c_j < \frac{\kappa^*}{k}$ By the induction hypothesis, $s_j \leq \frac{\kappa^*}{k} \Upsilon_j$. Since $c_j + p_j + s_j + \frac{\kappa^*}{k} \mathrm{rec}_j \geq \kappa^*$, it follows that $p_j > \frac{\kappa^*}{k}(k - 1 - \mathrm{rec}_j - \Upsilon_j)$, which implies $u_{aj}x_{aj} \leq \kappa^* - p_j < \frac{\kappa^*}{k}(1 + \mathrm{rec}_j + \Upsilon_j)$. The utility stolen from a by agents in T_i is thus bounded by $s_a \leq \sum_{j=1}^{r} \frac{\kappa^*}{k}(1 + \mathrm{rec}_j + \Upsilon_j) = \frac{\kappa^*}{k} \Upsilon_a$, completing the proof. □

C Other Results

Proof of Corollary 1. Let π^* be an optimal solution to the original instance. Use the flow algorithm given above for the case that all utilities lie in $\{0, 1\}$ with the modified utilities $u'_{ag} = \min\{u_{ag}, 1\}$. Let the resulting output be π. Suppose the minimum utility received by any agent under π^* is V, i.e. $\min_a u(a, \pi^*(a)) = V$. Then $\min_a u'(a, \pi^*(a)) \geq V/x$, and so by the optimality of π (in the modified instance), π is a V/x allocation in the modified instance. It immediately follows that π is a V/x allocation in the original instance, and thus an x approximation. □

Proof of Lemma 7. Consider a flow network with source s, sink t, edges (s, a) of capacity V to for each agent a, edges (g, t) of unit capacity for each good g, and edges $\{(a, g) | u_{ag} = 1\}$ of unit capacity. Since there exists an integral maximum flow, there is an V allocation iff there is a flow of value nV. Clearly, a V allocation can be easily obtained from a flow of value nV. So suppose by way of contradiction that $(\forall I \subseteq \mathcal{A},\ I \neq \emptyset)(f(I) \geq V|I|)$ and there is a cut $(S, \bar{S})$ of capacity $\mathtt{cap}(S, \bar{S}) < nV$. Let $\mathcal{A}_S = \mathcal{A} \cap S$. Then for each $a \notin \mathcal{A}_S$, $(s, a) \in \partial S$ and contributes V to the cut's capacity. Additionally, the agents in $\mathcal{A}_S$ must contribute at least $f(\mathcal{A}_S)$ to the cut's capacity. Thus $\mathtt{cap}(S, \bar{S}) < nV$ implies $|\mathcal{A} - \mathcal{A}_S|V + f(\mathcal{A}_S) < nV$, and so $f(\mathcal{A}_S) < |\mathcal{A}_S|V$, a contradiction. □

Proof of Lemma 10. Let M be any max-min fair matching, and π^* be the optimal allocation in the original (submodular) instance. Suppose by way of contradiction that the above claim is false. By submodularity, for each agent a, and set of goods S $u(a, S) \leq \sum_{g \in S} u(a, g)$. Label the agents such that agent i is the i^{th} worst off agent under π^*. For each i, $\vec{U}_i(\pi^*) \leq \sum_{g \in \pi^*(i)} u(i, g)$, and so $\max_{g \in \pi^*(i)}\{u(i, g)\} \geq \frac{1}{|\pi^*(i)|}\vec{U}_i(\pi^*)$. In any non-trivial instance, every agent receives at least one good in π^*, so no agent receives more than $m - n + 1$ goods. Let $V_M = \vec{U}_1(M)$ be the minimum utility received by any agent under M. Each agent receives utility at least $(m-n+1)V_M+\epsilon$ under π^*, for some $\epsilon > 0$. It follows that for each agent i, $\max_{g\in\pi^*(i)}\{u(i,g)\} \geq \frac{(m-n+1)V_M+\epsilon}{m-n+1} > V_M$. This contradicts the max-min fairness of M, since some matching $M' \subseteq \{(a,g)|g = \arg\max_{g\in\pi^*(a)}\{u(a,g)\}\}$ obtains a higher minimum utility. □

Proof of Theorem 8. Note that if M is strongly max-min fair with utilities u_{ag}, then M is strongly max-min fair with utilities $u'_{ag} = \phi(u_{ag})$; The total ordering on the utilities u_{ag}, and not their values, determine what the strongly max-min fair allocations are. Thus WLOG we may consider functions f from $\{1,2,\ldots,n^2\}$ to $\mathbb{N}$. We will show that there exists an $x \leq n^2$ such that $f(x) \geq n^{n^2-2n-1}$.

For fixed $a,b,c,n \in \mathbb{N}$, such that $0 < a \leq n^2-2n+1$ we construct instance $\mathcal{I}_n(a,b,c)$ as follows. Create n agents $\mathcal{A} = \{1,2,\ldots,n\}$, and n goods $\mathcal{G} = \{1,2,\ldots,n\}$. Let $u_{11} = a$, $u_{ii} = c$ for all $i > 1$, and $u_{ij} = b$ if $j = (i+1) \mod (n)$. Set the remaining utilities u_{ij} to be in $\{1,2,\ldots,a-1\}$, such that for each $v \in \{1,2,\ldots,a-1\}$, $u_{ij} = v$ for some (i,j) pair. This is possible since $a-1 \leq n^2-2n$. Note that applying ϕ to the utilities will leave $\mathcal{I}_n(a,b,c)$ unaltered if $c = b+1 = a+2$. Suppose further that $a < b < c \leq n^2-2n+3$. Let $M' = \{(i,(i+1) \mod (n))|1 \leq i \leq n\}$, $M = \{(i,i)|1 \leq i \leq n\}$. Note that M is the unique strongly max-min fair matching, and thus must be assigned maximum weight. In particular, $w(M) > w(M')$. Observe $w(M) = nf(b)$ and $w(M') = f(a)+(n-1)f(c)$. Thus $nf(b) > f(a)+(n-1)f(c)$, which implies $f(c)-f(a) > n(f(c)-f(b))$. Setting $b = a+1$ yields $f(c)-f(a) > n(f(c)-f(a+1))$. It follows that for any $k < c$

$$f(c)-f(1) > n(f(c)-f(2)) > n^2(f(c)-f(3)) > \ldots > n^k(f(c)-f(k+1))$$

and in particular, $f(c)-f(1) > n^{c-2}(f(c)-f(c-1))$. Clearly, f must be strictly increasing in the range $[1,n^2-2n]$ (consider instance $\mathcal{I}_n(x,x-1,x)$ with $x \leq n^2-2n$ to see why), and so $f(c)-f(c-1) \geq 1$. It follows that $f(c) > n^{c-2}$ if $3 \leq c \leq n^2-2n+3$, and so $f(n^2-2n+3) \geq n^{n^2-2n+1}$. □