The Complexity of Computing Maximin Share Allocations on Graphs

Gianluigi Greco, Francesco Scarcello
University of Calabria
87036 Rende (CS), Italy
{gianluigi.greco, francesco.scarcello}@unical.it

Abstract

Maximin share is a compelling notion of fairness proposed by Buddish as a relaxation of more traditional concepts for fair allocations of indivisible goods. In this paper we consider this notion within a setting where bundles of goods must induce connected subsets over an underlying graph. This setting received much attention in earlier literature, and our study answers a number of questions that were left open. First, we show that computing maximin share allocations is $\mathrm{F}\Delta_2^{\mathrm{P}}$-complete, even when focusing on *consistent* scenarios, that is, where such allocations are a-priori guaranteed to exist. Moreover, the problem remains intractable if all agents have the same type, i.e., have the same utility functions, and if either the values returned by the utility functions are polynomially bounded, or the underlying graphs have a low degree of cyclicity (more precisely, have bounded treewidth). However, if these conditions hold all together, then computing maximin share allocations (or checking that none exists) becomes tractable. The result is established via machineries based on logspace alternating machines that use partial representations of connected bundles, which are interesting in their own.

Introduction

Since the late forties, fair allocation of indivisible goods has been attracting much attention in the areas of mathematics, economics, and political science (Steihaus 1948; Brams and Taylor 1996). More recently, due to the intriguing computational and algorithmic questions that naturally arise within this context, fair division gained popularity in the AI community too—see, e.g., (Bouveret et al. 2016; Lang and Rothe 2016). The setting is very simple, but has a tremendously wide spectrum of applicability: we are given a set N of agents expressing some preferences over a set V of indivisible goods, and we want to allocate these goods to the agents in a way that is perceived as a "fair" one. For instance, we might require that no agent *envies* other agents.

Actually, most of the well-established notions of fairness, such as *envy-freeness* (or *equitability* or *proportionality*, just to name two further), are rather demanding in practice, so that defining meaningful relaxations is a very active topic of research. A noticeable example of a weaker (Bouveret and Lemaître 2016) but still very compelling notion is the *maximin share* proposed by Budish (2011). The idea is that every agent $i \in N$ must get a bundle $\mathbf{B}_i \subseteq V$ of goods for which she gets an utility that is not worse than the utility she could guarantee to herself if asked to divide the set of goods in $|N|$ bundles and then to keep the worst one. In fact, there are experimental and theoretical evidences showing that maximin share allocations can be singled out in many relevant settings (Kurokawa, Procaccia, and Wang 2016; Bouveret and Lemaître 2016); moreover, while their existence cannot be guaranteed in general (Procaccia and Wang 2014), allocations that "nearly satisfy" the maximin share criterion always exist and can be computed efficiently (Procaccia and Wang 2014; Amanatidis et al. 2017; Barman and Krishna Murthy 2017; Ghodsi et al. 2018; Nguyen, Nguyen, and Rothe 2017; Farhadi et al. 2019).

In this paper, we study maximin share allocations in the setting proposed by Bouveret et al. (2017), where bundles are subject to constraints captured by a connectivity condition over an underlying graph. More precisely, the nodes of the graph are the available goods and a bundle is allowed if the subgraph induced by such goods/nodes is connected. This condition emerges in many applications, often in presence of spatial constraints (e.g., when dividing offices among research groups) or temporal constraints (when goods represent time slots).

Fair allocation under connectivity constraints is receiving growing attention in the literature (Suksompong 2017; Lonc and Truszczynski 2018; Bouveret, Cechlárová, and Lesca 2018; Igarashi and Peters 2018; Bilò et al. 2019). Nevertheless, our knowledge is still quite partial when focusing on maximin share allocations: The problem of deciding whether a maximin share allocation exists is known to be NP-hard, and to belong to the complexity class Δ_2^{P} (Lonc and Truszczynski 2018). Moreover, computing such allocations is known to be tractable on trees (Bouveret et al. 2017) and on simple cycles (Lonc and Truszczynski 2018), but it was open whether it remains tractable on classes of graphs having more general structural properties, in particular for those classes of graphs having a limited degree of cyclicity—as it can be formalized by the notion of *bounded treewidth* (Robertson and Seymour 1986).

In this paper, we fill these gaps by giving a clear picture of the complexity of maximin share allocations under connectivity constraints, and by identifying islands of tractability when the degree of cyclicity is small. In particular,

- We first focus on *consistent* scenarios for which maximin share allocations are a-priori guaranteed to exist[1]. Rather surprisingly, we show that even on these scenarios (where the decision problem is trivial) computing a maximin share allocation is intractable, in fact complete for the class $\mathrm{F}\Delta_2^{\mathrm{P}}$—the natural counterpart of Δ_2^{P} on computation problems. The proof evidences that the main source of complexity lies in computing the minimum utility that each agent must guarantee to herself.
- Motivated by the above observation, we then embark on the study of more restricted settings. It turns out that computing maximin share allocations remains intractable even if there is one agent type only, i.e., all agents have the same utility functions, and if either *(i)* the values returned by the utility functions are polynomially bounded (equivalently, if they are given in unary notation), or *(ii)* the underlying graph has treewidth 2. In particular, point *(ii)* evidences that the tractability result on cycles (Lonc and Truszczynski 2018) cannot be extended to the more general class of those graphs having treewidth 2 (that is, having the same degree of cyclicity as the simple cycles, according to the treewidth measure).
- On the positive side, we are able to identify an island of tractability for instances whose underlying graphs are much more general than cycles: computing maximin share allocations is feasible in polynomial time for allocation scenarios with a bounded number of agent types, smooth utility functions, and bounded treewidth graphs.

Formal Framework and Preliminaries

Maximin Share Allocations. An *allocation scenario* is a tuple $\sigma = (N, G, \{u_i\}_{i\in N})$ where N is a set of agents, $G = (V, E)$ is a graph whose nodes in V are the available goods and, for each $i \in N$, u_i is the *utility function* of agent i, which maps each good $v \in V$ to a non-negative rational number $u_i(v)$. Utility functions are naturally extended to subsets of V by assuming *additivity*; to this end, we define $u_i(\emptyset) = 0$ for each $i \in N$. In the following, agents with the same utility function are said to have the same *type*. For any natural number h, denote by $[h]$ the set $\{1, \ldots, h\}$.

An *allocation* for σ is a tuple $\mathbf{B} = (\mathbf{B}_1, ..., \mathbf{B}_n)$ whose components associate each agent $i \in N$ with a *bundle* $\mathbf{B}_i \subseteq V$, i.e., with a set of goods such that the subgraph of G induced by them is connected. Moreover, since goods are indivisible, such bundles are required to be disjoint, that is, $\mathbf{B}_i \cap \mathbf{B}_{i'} = \emptyset$ for each pair of distinct agents i and i'.

The *maximin share* of agent $i \in N$ w.r.t. σ is defined as the value

$$\mathsf{mms}_\sigma(i) = \max_{\mathbf{B}} \min_{j\in N} u_i(\mathbf{B}_j)$$

[1]E.g., if all agents have the same utility, then a maximin share allocation is always guaranteed to exist.

An allocation $\mathbf{B}$ is a *maximin share* allocation if, for each $i \in N$, it holds that $u_i(\mathbf{B}_i) \geq \mathsf{mms}_\sigma(i)$. Note that maximin share allocations are not always guaranteed to exist (Lonc and Truszczynski 2018). Whenever a maximin share allocation exists for σ, then we say that σ is a *consistent* scenario.

Problem of Interest. In this paper we study the C-MMSA problem of computing a maximin share allocation (or checking that none exists). We assume a standard encoding for allocation scenarios σ. In particular, for each agent $i \in N$, her utility function u_i is encoded by explicitly listing all possible goods v with the associated value $u_i(v)$ represented in fractional form (and with binary notation). Therefore, the size of the encoding $||\sigma||$ is such that $||\sigma|| = O((|V|\log(|V|))^2 + |N| \times |V| \times M_\sigma)$, where M_σ is the maximum size over the encodings of the values returned by the utility functions. A class $\mathcal{C}$ of allocation scenarios has *smooth* utility functions if their output values are polynomially bounded, that is, $M_\sigma = O(\log(|N| + |V|))$, for each $\sigma \in \mathcal{C}$.[2]

Computational Complexity. Recall that an NP *metric Turing machine* MT is a polynomial-time nondeterministic Turing machine that, on every computation branch, halts with the binary encoding of a number on its output tape. The output of MT is the maximum number over its computations. The class OptP contains all integer functions that are computable by an NP metric Turing machine, and $\mathrm{OptP}[O(\log n)]$ is the subclass of OptP containing all functions f whose output values $f(x)$ have $O(\log n)$ bits, where n is the size of the input x. Then, $\mathrm{FNP}//\mathrm{OptP}$ (resp., $\mathrm{FNP}//\mathrm{OptP}[O(\log n)]$) is the class of all partial multi-valued functions g computed by polynomial-time nondeterministic Turing machines T such that, for every x, $g(x) = T(x \cdot h(x))$, where $\cdot$ denotes the concatenation operator, and h is a function in OptP (resp., $\mathrm{OptP}[O(\log n)]$) (Chen and Toda 1995). Note that $\mathrm{F}\Delta_2^{\mathrm{P}}$ can be viewed as the functional version of the complexity class Δ_2^{P}, hence consisting of all problems for which a solution can be computed in polynomial time by invoking with unitary cost an NP oracle. Observe that all functions in $\mathrm{F}\Delta_2^{\mathrm{P}}$ belong to $\mathrm{FNP}//\mathrm{OptP}$, and for each multi-valued function g in $\mathrm{FNP}//\mathrm{OptP}$, its single-valued refinements are in $\mathrm{F}\Delta_2^{\mathrm{P}}$ (Krentel 1988).

Tree decompositions. In our fine-grained complexity analysis, we study the C-MMSA problem on classes of quasi-acyclic graphs, as they are formalized by the notion of treewidth (Robertson and Seymour 1986). We recall that a *tree decomposition* of a graph $G = (V, E)$ is a pair $\langle T, \chi\rangle$, where T is a tree, and χ is a labeling function assigning to each vertex p in T a set of nodes $\chi(p) \subseteq V$, such that the following conditions are satisfied: (1) for each node $x \in V$, there exists p in T such that $x \in \chi(p)$; (2) for each edge $\{x, y\} \in E$, there exists p in T such that $\{x, y\} \subseteq \chi(p)$; and, (3) for each node $x \in V$, the subgraph of T induced by all vertices p such that $x \in \chi(p)$ is connected.

The *width* of $\langle T, \chi\rangle$ is the number $\max_{p\in T}(|\chi(p)| - 1)$. The *treewidth* of G, denoted by $tw(G)$, is the minimum

[2]All results we prove for smooth functions hold over scenarios where utility functions are encoded with unary notation, too.

width over all its decompositions. Treewidth is a generalization of acyclicity: G is acyclic if, and only if, $tw(G) = 1$.

Computing Maximin Share Allocations

In this section, we start our complexity analysis by showing that C-MMSA is complete for $\mathrm{F}\Delta_2^{\mathrm{P}}$. Rather surprisingly, the hardness result holds even on scenarios that are consistent—where the problem of deciding the existence of maximin share allocations trivializes. This is a remarkable result, since our hardness result on consistent scenarios matches the upper bound for the decision problem (membership in Δ_2^{P}) that is known to hold over scenarios that are not necessarily consistent (Lonc and Truszczynski 2018).

The analysis of C-MMSA is eventually completed on scenarios where agents have smooth utility functions.

Arbitrary Utility Functions

We first point out that C-MMSA belongs to the class $\mathrm{F}\Delta_2^{\mathrm{P}}$, the functional version of Δ_2^{P}.

Theorem 1 C-MMSA *belongs to* $\mathrm{F}\Delta_2^{\mathrm{P}}$.

Proof Sketch. The maximin share values can be computed by using the algorithm described in the proof that the problem of deciding whether there exists some maximin share allocation is in Δ_2^{P} (Lonc and Truszczynski 2018). Once the maximin share values have been computed, we can compute a maximin share allocation by using a self-reducibility argument on the NP problem of deciding whether there is an allocation with the given share values. □

We next show that C-MMSA is also hard for $\mathrm{F}\Delta_2^{\mathrm{P}}$ even on scenarios that are consistent, by exhibiting an involved reduction of the prototypical $\mathrm{F}\Delta_2^{\mathrm{P}}$-complete LEX-SAT problem (Krentel 1988): given a Boolean formula $\Phi = c_1 \wedge \cdots \wedge c_m$ in conjunctive normal form over the set $\{\alpha_1, ..., \alpha_n\}$ of variables, compute the *lexicographically maximum satisfying assignment*, with variables being ordered by their indices, or return -1 if Φ is not satisfiable. Equivalently, this problem can be defined by associating each variable α_h with a weight $w(\alpha_h) = 2^{n-h}$ and, for each truth assignment τ for Φ, defining its weight $w(\tau)$ as the sum of all weights $w(\alpha_h)$ of those variables α_h such that $\tau(\alpha_h) = \texttt{true}$. Accordingly, LEX-SAT is the problem of computing the satisfying assignment for Φ having the maximum possible weight (or -1, if Φ is not satisfiable). We first show the simple fact that this problem remains intractable even when Φ is satisfiable.

Lemma 2 LEX-SAT *is* $\mathrm{F}\Delta_2^{\mathrm{P}}$*-complete even if restricted on classes of satisfiable Boolean formulas.*

Proof Sketch. Let $\Phi = c_1 \wedge \cdots \wedge c_m$ be a formula defined over the variables in $\{\alpha_1, ..., \alpha_n\}$, and consider the formula $\Phi' = (\neg\alpha_{n+1} \vee c_1) \wedge \cdots \wedge (\neg\alpha_{n+1} \vee c_m) \wedge \bigwedge_{h=1}^{n}(\alpha_{n+1} \vee \neg\alpha_h)$ defined over the variables in $\{\alpha_1, ..., \alpha_{n+1}\}$. If Φ is not satisfiable, then the only satisfying assignment for Φ' is the one, say τ'_0, where every variable evaluates to `false`; note that $w(\tau'_0) = 0$. Otherwise, Φ is satisfied by the assignment τ if, and only if, Φ' is satisfied by the assignment τ' such that $\tau'(\alpha_{n+1}) = \texttt{true}$ and $\tau'(\alpha_h) = \tau(\alpha_h)$, for each $h \in \{1, ..., n\}$. Note that, in this latter case, we have

$$w(\tau') = \sum_{h \in \{1,...,n+1\} | \tau(\alpha_h) = \texttt{true}} 2^{n+1-h} = 2 \times w(\tau) + 1.$$

Hence, given the answer to LEX-SAT on input Φ' we can immediately return the answer to LEX-SAT on input Φ. □

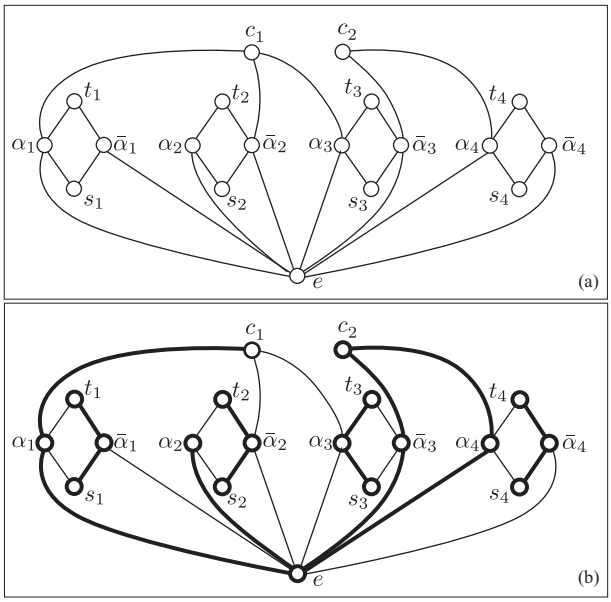

Figure 1: Reduction in the proof of Theorem 3: (a) The graph built for the formula $\Phi = c_1 \wedge c_2$ where $c_1 = \alpha_1 \vee \neg\alpha_2 \vee \alpha_3$ and $c_2 = \neg\alpha_3 \vee \alpha_4$; (b) The allocation $\mathbf{B}[\tau]$ (represented by evidencing the connected components corresponding to the bundles) such that τ is the assignment where α_3 is the only variable evaluating to false.

Theorem 3 C-MMSA *is* $\mathrm{F}\Delta_2^{\mathrm{P}}$*-hard, even on consistent scenarios with only one type of agents.*

Proof Sketch. Let $\Phi = c_1 \wedge \cdots \wedge c_m$ be a *satisfiable* Boolean formula over the set $\{\alpha_1, ..., \alpha_n\}$ of variables. Based on Φ, we build in polynomial time the allocation scenario $\sigma[\Phi] = (N, G, \{u_i\}_{i \in N})$ where $N = \{1, ..., n+1\}$ and where $G = (V, E)$ and the utility functions are defined as follows.

The set of goods is given by $V = \{s_h, t_h, \alpha_h, \bar{\alpha}_h \mid h \in \{1, ..., n\}\} \cup \{c_j \mid j \in \{1, ..., m\}\} \cup \{e\}$ and edges in E are such that (see Figure 1, for an illustration): for each $h \in \{1, ..., n\}$, the edges $\{s_h, \alpha_h\}$, $\{s_h, \bar{\alpha}_h\}$, $\{\alpha_h, t_h\}$, and $\{\bar{\alpha}_h, t_h\}$ are in E; for each $j \in \{1, ..., m\}$ and for each variable α_h occurring (either positively or negated) in c_j, both edges $\{\alpha_h, e\}$ and $\{\bar{\alpha}_h, e\}$ are in E; for each $j \in \{1, ..., m\}$ and for each variable α_h occurring positively in c_j, the edge $\{\alpha_h, c_j\}$ is in E; for each $j \in \{1, ..., m\}$ and for each variable α_h occurring negated in c_j, the edge $\{\bar{\alpha}_h, c_j\}$ is in E; and, no further edge is in E.

Finally, all agents are defined on the same utility function, so that it is immediate to check that maximin share

allocations are guaranteed to exist. In particular, for each $i \in N$, $u_i = u$ holds with u being the utility function such that: $u(s_h) = M + \frac{m}{2}(2^{n+1} + 2n) + 2^n + \frac{n}{2}$, for each $h \in \{1, ..., n\}$; $u(t_h) = M + \frac{m}{2}(2^{n+1} + 2n) + 2^n + \frac{n}{2}$, for each $h \in \{1, ..., n\}$; $u(\alpha_h) = w(\alpha_h) + 1$, for each $h \in \{1, ..., n\}$; $u(\bar{\alpha}_h) = 1$, for each $h \in \{1, ..., n\}$; $u(c_j) = 2^{n+1} + 2n$, for each $j \in \{1, ..., m\}$; and, $u(e) = 2M$, where $M = 4nm(2^{n+1}+2n)$. Note that $M > u(V) - 2nM - 2M$.

We now state some crucial properties of the construction. To this end, for each assignment τ for Φ, let us define $\mathbf{B}[\tau]$ as the tuple whose $n + 1$ elements are defined as follows:

- $\mathbf{B}[\tau]_h = \{t_h, s_h\} \cup X_h$ where X_h is either the set $\{\alpha_h\}$ or the set $\{\bar{\alpha}_h\}$ depending on whether $\tau(\alpha_h) = \texttt{false}$ or $\tau(\alpha_h) = \texttt{true}$, respectively.
- $\mathbf{B}[\tau]_{n+1} = V \setminus \bigcup_{h=1}^{n} \mathbf{B}[\tau]_h$.

Property P1 *If τ is an assignment that satisfies Φ, then $\mathbf{B}[\tau]$ is an allocation for $\sigma[\Phi]$.*

Proof of P1. By construction, the elements of $\mathbf{B}[\tau]$ are pairwise-disjoint subsets of goods. Moreover, for each $h \in \{1, ..., n\}$, it is immediate to check that the subgraph of $G = (V, E)$ induced by the goods $\mathbf{B}[\tau]_h$ is connected. To conclude, we show that the subgraph of G induced by the nodes in $\mathbf{B}[\tau]_{n+1}$ is connected, too. Indeed, the subgraph induced by the nodes in $\{e, \alpha_1, ..., \alpha_n, \bar{\alpha}_1, ..., \bar{\alpha}_n\} \cap \mathbf{B}[\tau]_{n+1}$ is clearly connected. Consider then a node c_j, for $j \in \{1, ..., m\}$. Since τ is a satisfying assignment, there is a literal occurring in it, say α_h or $\neg\alpha_h$, such that $\tau(\alpha_h) = \texttt{true}$ or $\tau(\alpha_h) = \texttt{false}$, respectively. In the former case, c_j is connected with the node α_h, which belongs to $\mathbf{B}[\tau]_{n+1}$ (as $\mathbf{B}[\tau]_h$ includes instead $\bar{\alpha}_h$, by construction). In the latter case, c_j is connected with $\bar{\alpha}_h$, which belongs to $\mathbf{B}[\tau]_{n+1}$ (as $\mathbf{B}[\tau]_h$ includes α_h). $\diamond$

Property P2 *If τ is an assignment that satisfies Φ, then $u(\mathbf{B}[\tau]_{n+1}) = 2M + m(2^{n+1} + 2n) + n + w(\tau)$. Moreover, $u(\mathbf{B}[\tau]_h) \geq u(\mathbf{B}[\tau]_{n+1})$, for each $h \in \{1, ..., n\}$.*

Proof of P2. Note that $\mathbf{B}[\tau]_{n+1} = \{e\} \cup \{c_1, ..., c_m\} \cup \{\alpha_h \mid \tau(\alpha_h) = \texttt{true}\} \cup \{\bar{\alpha}_h \mid \tau(\alpha_h) = \texttt{false}\}$. The fact that $u(\mathbf{B}[\tau]_{n+1}) = 2M + m(2^{n+1} + 2n) + n + w(\tau)$ then follows since $u(e) = 2M$, $u(c_i) = 2^{n+1} + 2n$ for each $i \in \{1, ..., m\}$, and $u(\alpha_h) = w(\alpha_h) + 1$ and $u(\bar{\alpha}_h) = 1$. Moreover, note that $u(\mathbf{B}[\tau]_h) \geq 2M + m(2^{n+1}+2n) + n + 2^{n+1}$, for each $h \in \{1, ..., n\}$, so that $u(\mathbf{B}[\tau]_h) \geq u(\mathbf{B}[\tau]_{n+1})$ derives because $w(\tau) < 2^{n+1}$. $\diamond$

Property P3 *From any maximin share allocation, a lexicographically maximum satisfying assignment for Φ can be obtained in polynomial time.*

Proof of P3. Let $\mathbf{B}^*$ be a maximin share allocation, and observe that $u(\mathbf{B}^*_i) \geq 2M + m(2^{n+1}+2n) + n + w^*$, for each $i \in N$ and where w^* is the weight of a lexicographically maximum satisfying assignment. Indeed, Properties **P1** and **P2** guarantee that such a best allocation has at least the above value for every bundle in $\mathbf{B}^*$.

Note that $\{v \in V \mid u(v) \geq M\} = \{e\} \cup \bigcup_{h=1}^{n} \{s_h, t_h\}$. In fact, since $M > u(V) - 2nM - 2M$ and since—in particular—we must have $u(\mathbf{B}^*_i) \geq 2M$, for each $i \in N$, then we conclude that one of the bundles, say $\mathbf{B}^*_{n+1}$, must contain e, and any other bundle must contain precisely two nodes in $\{s_h, t_h \mid h \in \{1, ..., n\}\}$. Furthermore, note that $u(c_j) = 2^{n+1} + 2n > u(\{\alpha_1, ..., \alpha_n, \bar{\alpha}_1, ..., \bar{\alpha}_n\})$, for each $j \in \{1, ..., m\}$. Therefore, from the above properties and the fact that $u(\mathbf{B}^*_{n+1}) \geq 2M + m(2^{n+1} + 2n) + n + w^*$ we can entail $\mathbf{B}^*_{n+1} \supseteq \{e, c_1, ..., c_m\}$. Consider now any agent $i \in \{1, ..., n\}$, and recall that the subgraph induced by the nodes in $\mathbf{B}^*_i$ is connected. Combined with the above fact, this implies the existence of some index $h(i) \in \{1, ..., n\}$ such that both $s_{h(i)}$ and $t_{h(i)}$ occur in $\mathbf{B}^*_i$ and $\{\alpha_{h(i)}, \bar{\alpha}_{h(i)}\} \cap \mathbf{B}^*_i \neq \emptyset$. Moreover, note that either $\alpha_{h(i)}$ or $\bar{\alpha}_{h(i)}$ must belong to $\mathbf{B}^*_{n+1}$, because $\mathbf{B}^*$ is a maximin share allocation, $u(\mathbf{B}^*_{n+1})$ is less than the values of the other bundles, and $u(\mathbf{B}^*_{n+1}) \geq 2M + m(2^{n+1}+2n) + n + w^*$ (so that every Boolean variable must contribute to this worst bundle with a weight).

Then, define a truth assignment τ as follows: for each $i \in \{1, ..., n\}$, $\tau(\alpha_{h(i)}) = \texttt{true}$ if $\alpha_{h(i)} \in \mathbf{B}^*_{n+1}$; otherwise, $\tau(\alpha_{h(i)}) = \texttt{false}$. We claim that τ is satisfying. Indeed, since the subgraph of G induced by the nodes in $\mathbf{B}^*_{n+1}$ is connected, then for each node c_j with $j \in \{1, ..., m\}$, $\mathbf{B}^*_{n+1}$ must contain at least one of her neighbors, i.e., a node having the form $\alpha_{h(i)}$ (resp., $\bar{\alpha}_{h(i)}$) if the variable $\alpha_{h(i)}$ occurs positively (resp., negated) in c_j. If $\alpha_{h(i)}$ is in $\mathbf{B}^*_{n+1}$, $\tau(\alpha_h) = \texttt{true}$, by construction, and if $\bar{\alpha}_{h(i)}$ is in $\mathbf{B}^*_{n+1}$, then $\tau(\alpha_h) = \texttt{false}$. In both cases, τ satisfies the clause c_j.

To conclude, consider the allocation $\mathbf{B}[\tau]$ and observe that it is the same as $\mathbf{B}^*$, modulo a permutation of the bundles assigned to agents in $\{1, ..., n\}$, so that $\sum_{i \in \{1,...,n\}} u(\mathbf{B}[\tau]_i) = \sum_{i \in \{1,...,n\}} u(\mathbf{B}^*_i)$ and $u(\mathbf{B}[\tau]_{n+1}) = 2M + m(2^{n+1}+2n) + n + w(\tau) = u(\mathbf{B}^*_{n+1})$, which entails $w(\tau) = w^*$. $\diamond$

By latter property, we immediately derive that from any solution to C-MMSA on input $\sigma[\Phi]$, we get a solution to LEX-SAT on input Φ. Note that $\sigma[\Phi]$ is a consistent scenario, since it contains only one type of agent. The desired complexity result then follows by Lemma 2. $\square$

Smooth Utility Functions

The careful reader might have noticed that, in the proof of Theorem 3, we have exploited utility functions taking values that are exponential with respect to the number of agents and goods. However, values used in practice might be not that large, and it makes sense to consider the restriction of the general problem to those instances having *smooth* utility functions. In this section we show that the problem is in fact slightly easier when such functions are considered, but it remains intractable, precisely $\mathrm{FNP}//\mathrm{OptP}[O(\log n)]$-complete (Chen and Toda 1995).

Theorem 4 *On smooth utility functions,* C-MMSA *is $\mathrm{FNP}//\mathrm{OptP}[O(\log n)]$-complete. Hardness holds even on consistent scenarios with only one type of agents.*

Proof Sketch. *(Membership)* On smooth utility functions, we have polynomially many utility values and we can show that the maximin share of all agents can be computed by using an NP metric machine with $O(\log n)$ output bits. Given

the knowledge of such maximin shares, we can guess an allocation $\mathbf{B} = (\mathbf{B}_1, ..., \mathbf{B}_n)$, by subsequently verifying in polynomial time that, for each agent i, $u_i(\mathbf{B}_i) \geq \mathsf{mms}_\sigma(i)$ actually holds.

(Hardness) We exhibit a reduction of the problem $\mathcal{X}$-MAXIMAL MODEL, which is $\mathrm{FNP}//\mathrm{OptP}[O(\log n)]$-complete (Chen and Toda 1995): given a formula $\Phi = c_1 \wedge \cdots \wedge c_m$ over the set $\{\alpha_1, ..., \alpha_n\}$ of variables and a subset $\mathcal{X} \subseteq \{\alpha_1, ..., \alpha_n\}$, we are asked to compute a satisfying truth assignment τ whose $\mathcal{X}$-part is maximal. That is, there is no satisfying truth assignment τ' such that $\{\alpha_i \mid \tau'(\alpha_i) = \mathtt{true}\} \cap \mathcal{X} \supset \{\alpha_i \mid \tau(\alpha_i) = \mathtt{true}\} \cap \mathcal{X}$.

It is easy to see—and it has been already observed in the literature (Eiter et al. 2007)—that one can focus, w.l.o.g., on formulas Φ that are satisfiable. Hence, the reduction is identical to the one used in the proof of Theorem 3, provided that we set $w(\alpha_h) = 1$ (resp., $w(\alpha_h) = 0$) if $\alpha_h \in \mathcal{X}$ (resp., $\alpha_h \notin \mathcal{X}$), and that in the utility functions we replace each expression of the form 2^n (resp., 2^{n+1}) with n (resp., $2n$).

Indeed, with these adjustments, it can be checked that the whole proof still works (basically, we have just to consistently replace the above values in the proof). So, we get that any maximin share allocation corresponds to a satisfying assignment τ^* for Φ with a maximum number of variables from $\mathcal{X}$ evaluating to true (rather than to a lexicographically maximum satisfying assignment). Clearly, the satisfying assignment τ^* is also a solution to $\mathcal{X}$-MAXIMAL MODEL. □

Maximin Share Allocations on Quasi-Acyclic Graphs

Since C-MMSA is already known to be tractable when restricted to acyclic graphs (Bouveret et al. 2017), a natural avenue of research is to assess whether it remains tractable by considering proper generalizations of acyclicity, such as the notion of *bounded treewidth* (Robertson and Seymour 1986). This is a well-known measure of the degree of cyclicity of graphs—with acyclic graphs being all and only graphs having treewidth 1.

Hard Quasi-Acyclic Scenarios

In many reasoning problems arising in AI (Gottlob, Greco, and Scarcello 2014), tractability over acyclic structures usually extends to quasi-acyclic ones. A natural step in this direction is the polynomial time algorithm for *cycles* (whose treewidth is 2) exhibited in the literature for scenarios with a fixed number of agent types (Lonc and Truszczynski 2018). Unfortunately, the following result shows that, somehow surprisingly, moving from simple cycles to the larger class of those cyclic graphs having the same (minimum) treewidth 2, makes the problem intractable.

Theorem 5 C-MMSA *is* NP*-hard even on classes of instances having graphs G such that $tw(G) = 2$. Hardness holds even if there is a fixed number of agents (having the same type).*

Proof Sketch. The well-known NP-hard PARTITION problem (Garey and Johnson 1979) takes as input a multiset of positive integers $I = \{s_1, ..., s_\ell\}$ and asks whether it is possible to find two disjoint multisets S_1 and S_2 with $S_1 \cup S_2 = I$ and such that $\sum_{x \in S_1} x = \sum_{x \in S_2} x$. Assume, w.l.o.g., that $2M = \sum_{x \in I} x$ (that is, $\sum_{x \in I} x$ is even).

Given the multiset I, we build in polynomial time the allocation scenario $\sigma[I] = (N, G, \{u_i\}_{i \in N})$, with $G = (N, E)$, $N = \{1, 2\}$, and $V = \{w_1, w_2, v_1, ..., v_\ell\}$. In particular, the graph $G = (V, E)$ is such that $\{w_1, v_h\}$ and $\{w_2, v_h\}$ are in E, for each $h \in \{1, ..., \ell\}$; and no further edge is in E. Moreover, the two agents have the same utility function u, which is such that: $u(w_1) = u(w_2) = 0$ and $u(v_h) = s_h$, for each $h \in \{1, ..., \ell\}$. Note that $tw(G) = 2$.

We claim that: I is a "yes" instance to PARTITION if, and only if, there is a maximin share allocation $\mathbf{B}$ for $\sigma[I]$ with $u(\mathbf{B}_1) = u(\mathbf{B}_2) = M$.

(only-if part) Assume that $S_1 \subseteq I$ and $S_2 \subseteq I$ are two multisets with $S_1 \cup S_2 = I$ and $\sum_{v_h \in S_1} v_h = M$ and $\sum_{v_h \in S_2} v_h = M$. Consider the tuple $\mathbf{B}$ such that: $\mathbf{B}_1 = \{w_1\} \cup \{v_h | s_h \in S_1\}$ and $\mathbf{B}_2 = \{w_2\} \cup \{v_h | s_h \in S_2\}$. Note that $\mathbf{B}$ is an allocation with $u(\mathbf{B}_1) = u(\mathbf{B}_2) = M$. In particular, it is a maximin share allocation because $u(V) = 2M$.

(if part) Assume that a maximin share allocation, say $\mathbf{B}$, exists such that $u(\mathbf{B}_1) = u(\mathbf{B}_2) = M$. Since $u(V) = 2M$ and by the connectedness properties of the bundles, we can write, w.l.o.g., $\mathbf{B}_1 = \{w_1\} \cup V_1$ and $\mathbf{B}_2 = \{w_2\} \cup V_2$, where V_1 and V_2 are two subsets of $\{v_1, ..., v_\ell\}$ such that $\sum_{v_h \in V_1} u(v_h) = M$ and $\sum_{v_h \in V_2} u(v_h) = M$. Therefore, $S_1 = \{s_h \mid v_h \in V_1\}$ and $S_2 = \{s_h \mid v_h \in V_2\}$ witness that I is a "yes" instance to PARTITION. □

Islands of Tractability for Maximin Share Allocations via Bounded Treewidth

Note that in the proof of Theorem 5, the graph used in the encoding has bounded treewidth but utility functions are not smooth. To the contrary, from Theorem 4, we already know that smooth utility functions are intractable too, but the graph used in that reduction does not have bounded treewidth. Therefore, it is natural to ask what happens if we apply bounded treewidth with smooth utility functions. Our main result in this section is a polynomial-time algorithm that works for such instances. In particular, note that the result holds even if the number of agents is arbitrary, but the number of their distinct types is required to be small.

Theorem 6 *On classes of smooth allocation scenarios with a bounded number of agent types and having bounded treewidth,* C-MMSA *can be solved in polynomial time.*

Proof Idea. Let $\sigma = (N, G, u)$, with $G = (V, E)$, be a smooth allocation scenario with h different agent types. Let n_j, $\forall j \in [h]$, be the number of agents of type j, with $\sum_{j \in [h]} n_j = |N|$. Assume that $tw(G) \leq k$, for some fixed $k \geq 1$. Then we can compute a tree decomposition of the graph G having width at most k in linear time (Bodlaender 1993). Moreover, we can easily transform this decomposition into a tree decomposition $\langle T, \chi \rangle$ of G having the same width and such that T is a (rooted) full binary tree, that is, every node is either a leaf or has two children.

Input: $\sigma, M, \langle T, \chi \rangle$
Task: decide whether there is an allocation $\mathbf{B}$ such that, $\forall j \in [h]$, $u_a(\mathbf{B}_a) \geq M_j$, for each agent $a \in N$ with $type(a) = j$
Method:
let q be $root$ of T
CHECK$(q, \emptyset, \emptyset, \emptyset, \langle n_1, \ldots, n_h \rangle)$

Procedure CHECK$(q, \lambda_p, \omega_p, CT_p, rest)$
Begin Procedure
guess a (total) mapping $\lambda_q : \chi(q) \mapsto N$ assigning to agents the goods occurring at q
`Stop and Fail if` $\lambda_q(g) \neq \lambda_p(g)$, for some good g occurring in both mappings
let $new = AG_q \setminus AG_p$, and let $new_j = \{a \in AG_q \setminus AG_p \mid type(a) = j\}$ (the new type-j active agents introduced at q)
let $terminal = AG_p \setminus AG_q$ (the agents that are active at p and disappear at q)
for each agent $a \in terminal$, `Stop and Fail` **if**
$\omega_p[a] > 0$ (the goods assigned to a are not enough), or
$CT_p[a]$ contains more than one component (the bundle of goods assigned to a is not connected)
for each type $j \in [h]$ and **each** agent $a \in new_j$:
compute the minimum value to be assigned to a: $\omega_q[a] = M_j - \sum_{g \in goods_q[a]} u_a(g)$
guess the components-tree $CT_q[a]$ with the connected components of the induced subgraph $G[goods_q[a]]$
for each other agent $a' \in AG_q \setminus new$:
update the minimum value to be assigned to a': $\omega_q[a'] = \omega_p[a'] - \sum_{g \in (goods_q[a'] \setminus goods_p[a'])} u_{a'}(g)$
guess an update of the components-tree $CT_q[a']$ (to consider new goods and remove disappeared goods)
if q is a leaf of T **then**
if $rest_j \neq |new_j|$ for some $j \in [h]$, **then** `Stop and Fail` (there is some agent without any bundle of goods)
for each agent $a \in AG_q$, `Stop and Fail` **if**
$\omega_q[a] > 0$ (the goods assigned to a are not enough), or
$CT_q[a]$ contains more than one component (the bundle of goods assigned to a is not connected)
otherwise (q is not a leaf of T)
guess two arrays of numbers $rest'$ and $rest''$ such that $rest'_j + rest''_j = rest_j - |new_j|$, for each $j \in [h]$
guess two sets of remaining values ω'_q and ω''_q such that, $\forall a \in AG_q$, $\omega_q[a] = \omega'_q[a] + \omega''_q[a]$
guess a split $(CT'_q[a], CT''_q[a])$ of the components-tree of each $a \in AG_q$
Let ℓ and r be the left child and the right child of q in T
CHECK$(\ell, \lambda_q, \omega'_q, CT'_q, rest')$
CHECK$(r, \lambda_q, \omega''_q, CT''_q, rest'')$
End Procedure

Figure 2: Algorithm EXISTBUNDLES.

Our algorithm is based on a number of calls to a function called EXISTBUNDLES, which is reported in Figure 2. The function gets as its input the allocation scenario σ, a vector $\bar{M}$ of h rational numbers holding the minimum value M_j to be guaranteed to each agent having type $j \in [h]$, and the tree decomposition $\langle T, \chi \rangle$ of G. EXISTBUNDLES decides whether there is an allocation $\mathbf{B}$ such that, for each type $j \in [h]$, $u_a(\mathbf{B}_a) \geq M_j$, for every agent $a \in N$ having type j.

EXISTBUNDLES is a non-deterministic function that can be implemented on a logspace *Alternating Turing Machine (ATM)*, which entails that it can be implemented on a polynomial-time deterministic Turing machine, too. It is easy to obtain a (standard deterministic) dynamic-programming algorithm given an ATM implementation (Chandra, Kozen, and Stockmeyer 1981). However, inventing directly such a standard algorithm may be highly non-trivial, as in our case. Recall that an ATM is a non-deterministic machine having existential states and universal states, and accepts its input if there is an accepting computation tree (instead of an accepting path, as in standard Turing machines). The latter is a tree of machine descriptions, where all leaves hold accepting states, each description with an existential state has one child (encoding the non-deterministic choice performed at that point), and each universal state has some children (hence, subtrees of the computations tree). Whenever a logspace ATM is guaranteed to have computation trees having polynomial size for each "yes" input-instance, the problem solved by this machine belongs to the complexity class LogCFL, and it is thus parallelizable (Ruzzo 1980). We shall show that this is the case for EXISTBUNDLES, and this entails that such an accepting computation tree, which in our problem encodes the bundles assigned to agents, can be computed in the functional version of LogCFL (Gottlob, Leone, and Scarcello 2002). It follows that C-MMSA can be solved within the same complexity, and hence in polynomial time.

Theorem 6 is then established as C-MMSA can be solved in polynomial time by using EXISTBUNDLES as follows:

- We first compute the value $\mathsf{mms}_\sigma(j)$ for each agent type $j \in [h]$. This value can be computed by looking for the maximum share value that can be guaranteed to all agents in the modified allocation scenario, say σ_j, where there is only the one type j (Bouveret et al. 2017). This can be obtained by performing a logarithmic search with suitable

values M on the range $[0, u_a(V)]$ to be checked (where a is any type-j agent), by calling EXISTBUNDLES on the modified scenario σ_j (and with the tree decomposition $\langle T, \chi \rangle$). This computation requires clearly polynomial time and should be repeated for each type, hence a constant number of times.

- After the previous step, it is sufficient to call EXISTBUNDLES with the parameters σ and $\langle T, \chi \rangle$, and with the vector of values holding $\mathsf{mms}_\sigma(j)$ for each type $j \in [h]$.

In order to conclude the proof, we analyze EXISTBUNDLES and its properties.

The function EXISTBUNDLES: It is designed to run on an ATM that, intuitively, works top-down along the tree decomposition $\langle T, \chi \rangle$. We use the term "node" to refer to any node of the decomposition tree T, while we use the term "good" to refer to any node of the graph G. If $\xi : X \mapsto Y$ is a partial mapping from X to Y, we write $x \in \xi$ to mean any element x in the active domain of ξ, that is, such that $x \in X$ is actually mapped to some $\xi(x) \in Y$. Moreover, $\emptyset$ denotes the empty mapping (having an empty active domain).

At each node q of T, the ATM performs a number of non-deterministic choices by using its existential states. These choices are then checked on the subtrees rooted at the children of q by using the universal states. We describe the procedure as a high-level recursive procedure with the ability to perform non-deterministic *guess* operations. More precisely, when called with a node q as its parameter, the procedure CHECK starts *guessing* an assignment of goods $\lambda_q : \chi(q) \mapsto N$. Denote by AG_q the so-called *active agents* at q, that is, the set of agents $\{a \in N \mid \lambda_q(g) = a, \text{ for some } g \in \chi(q)\}$ to which the goods occurring at q are assigned. Denote by $goods_q[a]$ the set of goods assigned to such an agent a at q, i.e., the set $\{g \in \chi(q) \mid \lambda_q(g) = a\}$. Note that there are at most $k+1$ goods in $\chi(q)$, and thus at most $k+1$ active agents at q. These sets and the mapping can be stored with $O(\log n)$ bits (indeed, goods, agents, and nodes are encoded with suitable pointers to the input tape, or indices identifying such elements). The machine also stores for each active agent a the minimum remaining value $\omega_q[a]$ to be assigned to a: if $type(a) = j$, it is initially the minimum required value M_j, and it is then reduced by considering the values of the goods assigned to a along the tree. Before the recursive calls on the children of q in T, the ATM *guesses* two values $\omega'_q[a]$ and $\omega''_q[a]$, such that $\omega_q[a] = \omega'_q[a] + \omega''_q[a]$, that encode the values that remains to be assigned to a in the subtrees rooted at the children of q. Similarly, for each type j, it is stored the number of agents $rest_j$ to be dealt with in the subtree rooted at q; at the root, this number is n_j. When a new active agent of type j is introduced at some node, $rest_j$ is decremented. Before the recursive calls on the children of q in T the ATM *guesses* two values $rest'_j$ and $rest''_j$ holding the number of remaining agents of type j to be dealt with in the subtrees rooted at the children of q. At leaves, all these numbers should go to zero.

A difficult point in this non-deterministic logspace algorithm, is requiring that the bundle $\mathbf{B}_a$ of goods assigned to each agent a is connected, and that no good can be assigned to different agents during the algorithm, possibly at different non-adjacent nodes of T. Note that we do not have enough memory to store all the goods of a bundle, let alone the goods in the bundles of all agents. Consider a connected bundle of goods $\mathbf{B}_a$. From the properties of tree decompositions, it can be seen easily that the nodes of the decomposition tree T containing goods of $\mathbf{B}_a$ induce a connected subtree of T. However, the goods $goods_q[a]$ from $\mathbf{B}_a$ that occur at node q of this subtree can be goods that are not directly connected in the graph G. Because of the logarithmic-space constraint, we may have only a partial view of $\mathbf{B}_a$, and we cannot say anything about the connection property of $\mathbf{B}_a$.

The main ingredient here is to manage, at such a node q of the decomposition-tree, a logarithmic-space partial representation of the bundle assigned to each active agent a at that node, which we may call *components-tree* of a at q, and denoted by $CT_q[a]$. The vertices of $CT_q[a]$ encode a partition of the goods $goods_q[a]$ assigned to a at q, and are initially set to be the connected components of the subgraph $G[goods_q[a]]$ induced on G by these goods. Note that there are at most k such vertices, because the treewidth is k and at most $k+1$ goods occur at q. The edges (at most k) are *guessed* non-deterministically by the ATM, and encode paths among the connected components. Such paths include goods, still unknown at this point of the algorithm, that will be eventually found along the decomposition tree, because the bundle must be connected. We cannot expect to check such paths coming across the missing goods in every branch of the tree T where the agent a at hand is active. Rather, at any node q with two children ℓ and r, it is possible that some edges will be dealt with in the subtree rooted at ℓ and other edges in the subtree rooted at r. We thus use in the algorithm what we call a *split operation* to produce, non-deterministically, a suitable pair of components-tree $CT'_q[a]$ and $CT''_q[a]$ to be checked in the recursive calls.

Eventually, EXISTBUNDLES executes a recursive call for each child of q in order to check, over the subtrees rooted at ℓ and r, that the non-deterministic choices performed at q are actually correct. Observe that all the information needed at each call of EXISTBUNDLES can be encoded in $O(\log n)$. Moreover, it can be seen that computation trees have polynomial size for each "yes" input-instance, so that EXISTBUNDLES is in LogCFL, as required. □

Conclusion

We have depicted a clear picture of the complexity of computing maximin share allocations over graphs. We have shown that the problem is in general $\mathrm{F}\Delta_2^{\mathrm{P}}$-complete, and that it remains intractable if there is a small number of types, and if either the values returned by the utility functions are polynomially bounded, or the underlying graph has a low degree of cyclicity. However, if these conditions hold together, then the problem becomes solvable in polynomial time.

The techniques we have used to identify islands of tractability are rather elaborated. Interestingly, they can be smoothly applied to analyze settings where utilities can be negative, that is, when some of the items to be allocated

are actually *chores* (Aziz et al. 2017; Aziz, Caragiannis, and Igarashi 2018). Our research leaves open the question about whether bounded treewidth and smooth utility functions alone (i.e., without the bound on the number of types) are sufficient to guarantee tractability.

References

Amanatidis, G.; Markakis, E.; Nikzad, A.; and Saberi, A. 2017. Approximation algorithms for computing maximin share allocations. *ACM Trans. Algorithms* 13(4):52:1–52:28.

Aziz, H.; Rauchecker, G.; Schryen, G.; and Walsh, T. 2017. Algorithms for max-min share fair allocation of indivisible chores. In *Proc. of AAAI'17*, 335–341.

Aziz, H.; Caragiannis, I.; and Igarashi, A. 2018. Fair allocation of combinations of indivisible goods and chores. *arXiv preprint arXiv:1807.10684*.

Barman, S., and Krishna Murthy, S. K. 2017. Approximation algorithms for maximin fair division. In *Proc. of EC'17*, 647–664.

Bilò, V.; Caragiannis, I.; Flammini, M.; Igarashi, A.; Monaco, G.; Peters, D.; Vinci, C.; and Zwicker, W. S. 2019. Almost envy-free allocations with connected bundles. In *Proc. of ITCS'19*, 14:1–14:21.

Bodlaender, H. L. 1993. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proc. of STOC'93*, 226–234.

Bouveret, S., and Lemaître, M. 2016. Characterizing conflicts in fair division of indivisible goods using a scale of criteria. *Autonomous Agents and Multi-Agent Systems* 30(2):259–290.

Bouveret, S.; Chevaleyre, Y.; Maudet, N.; and Moulin, H. 2016. *Fair Allocation of Indivisible Goods*. Cambridge University Press. 284–310.

Bouveret, S.; Cechlárová, K.; Elkind, E.; Igarashi, A.; and Peters, D. 2017. Fair division of a graph. In *Proc. of IJCAI'17*, 135–141.

Bouveret, S.; Cechlárová, K.; and Lesca, J. 2018. Chore division on a graph. *Autonomous Agents and Multi-Agent Systems* 1–24.

Brams, S. J., and Taylor, A. D. 1996. *Fair division: From cake-cutting to dispute resolution*. Cambridge University Press.

Budish, E. 2011. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy* 119(6):1061–1103.

Chandra, A. K.; Kozen, D. C.; and Stockmeyer, L. J. 1981. Alternation. *J. ACM* 28(1):114–133.

Chen, Z., and Toda, S. 1995. The complexity of selecting maximal solutions. *Information and Computation* 119(2):231–239.

Eiter, T.; Erdem, E.; Faber, W.; and Senko, J. 2007. A logic-based approach to finding explanations for discrepancies in optimistic plan execution. *Fundamenta Informaticae* 79:25–69.

Farhadi, A.; Ghodsi, M.; Hajiaghayi, M. T.; Lahaie, S.; Pennock, D. M.; Seddighin, M.; Seddighin, S.; and Yami, H. 2019. Fair allocation of indivisible goods to asymmetric agents. *Journal of Artifcial Intelligence Reserach* 64:1–20.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Ghodsi, M.; Hajiaghayi, M.; Seddighin, M.; Seddighin, S.; and Yami, H. 2018. Fair allocation of indivisible goods: Improvements and generalizations. In *Proc. of EC'18*, 539–556.

Gottlob, G.; Greco, G.; and Scarcello, F. 2014. Treewidth and hypertree width. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press. 3–38.

Gottlob, G.; Leone, N.; and Scarcello, F. 2002. Computing LOGCFL certificates. *Theor. Comput. Sci.* 270(1-2):761–777.

Igarashi, A., and Peters, D. 2018. Pareto-optimal allocation of indivisible goods with connectivity constraints. *arXiv preprint arXiv:1811.04872*.

Krentel, M. W. 1988. The complexity of optimization problems. *Journal of Computer and System Sciences* 36(3):490–509.

Kurokawa, D.; Procaccia, A. D.; and Wang, J. 2016. When can the maximin share guarantee be guaranteed? In *Proc. of AAAI'16*, 523–529.

Lang, J., and Rothe, J. 2016. Fair division of indivisible goods. In *Economics and Computation*. Springer. 493–550.

Lonc, Z., and Truszczynski, M. 2018. Maximin share allocations on cycles. In *Proc. of IJCAI'18*, 410–416.

Nguyen, N.-T.; Nguyen, T. T.; and Rothe, J. 2017. Approximate solutions to max-min fair and proportionally fair allocations of indivisible goods. In *Proc. of AAMAS'17*, 262–271.

Procaccia, A. D., and Wang, J. 2014. Fair enough: Guaranteeing approximate maximin shares. In *Proc. of EC '14*, 675–692.

Robertson, N., and Seymour, P. 1986. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms* 7(3):309–322.

Ruzzo, W. L. 1980. Tree-size bounded alternation. *J. Comput. Syst. Sci.* 21(2):218–235.

Steihaus, H. 1948. The problem of fair division. *Econometrica* 16:101–104.

Suksompong, W. 2017. Fairly allocating contiguous blocks of indivisible items. In *Algorithmic Game Theory*, 333–344.