Reducing Leximin Fairness to Utilitarian Optimization

Eden Hartman[1], **Yonatan Aumann**[1], **Avinatan Hassidim**[1,2], **Erel Segal-Halevi**[3]

[1]Bar-Ilan University, Israel
[2]Google, Israel
[3]Ariel University, Israel
eden.r.hartman@gmail.com, aumann@cs.biu.ac.il, avinatan@cs.biu.ac.il, erelsgl@gmail.com

Abstract

Two prominent objectives in social choice are *utilitarian* - maximizing the sum of agents' utilities, and *leximin* - maximizing the smallest agent's utility, then the second-smallest, etc. Utilitarianism is typically computationally easier to attain but is generally viewed as less fair. This paper presents a general reduction scheme that, given a utilitarian solver, produces a distribution over states (deterministic outcomes) that is leximin in expectation. Importantly, the scheme is robust in the sense that, given an *approximate* utilitarian solver, it produces a lottery that is approximately-leximin (in expectation) - with the same approximation factor. We apply our scheme to several social choice problems: stochastic allocations of indivisible goods, giveaway lotteries, and fair lotteries for participatory budgeting.

Extended Version — https://arxiv.org/abs/2409.10395

1 Introduction

In social choice, the goal is to find the best choice for society, but 'best' can be defined in many ways. Two frequent, and often contrasting definitions are the *utilitarian best*, which focuses on maximizing the total welfare (i.e., the sum of utilities); and the *egalitarian best*, which focuses on maximizing the least utility. The *leximin best* generalizes the egalitarian one. It first aims to maximize the least utility; then, among all options that maximize the least utility, it chooses the one that maximizes the second-smallest utility, among these — the third-smallest utility, and so forth. Leximin is often the solution of choice in social choice applications, and frequently used (e.g., Freeman et al. (2019); Bei, Lu, and Suksompong (2022); Cheng et al. (2023); Flanigan et al. (2024)).

Calculating the Optimal Choice. Calculating a choice that maximizes utilitarian welfare is often easier than finding one that maximizes egalitarian welfare, while finding one that is leximin optimal is typically even more complex. For example, when allocating indivisible goods among agents with additive utilities, finding a choice (in this case, an allocation) that maximizes the utilitarian welfare can be done by greedily assigning each item to the agent who values it most. Finding an allocation that maximizes the egalitarian welfare, however, is NP-hard (Bansal and Sviridenko 2006), even in this relatively simple case.

In this paper, we show that knowing how to efficiently maximize the utilitarian welfare is sufficient in order to find a fair leximin solution.

Contributions. The core contribution of this paper is a general protocol that, when provided with a procedure for optimizing the utilitarian welfare (for a given problem), outputs a solution that optimizes the expected leximin welfare (for the same problem). By *expected* leximin we mean a distribution over deterministic solutions, for which the expectations of the players' utilities is leximin optimal. Crucially, our protocol *extends to approximations*, in the following sense: given an *approximate* solver for the utilitarian welfare, the protocol outputs a solutions that approximates the expected leximin optimal, and the same approximation factor is preserved. In all, with our protocol at hand, optimizing expected leximin welfare is no more difficult than optimizing utilitarian welfare.

We demonstrate the significance of this reduction by applying it to three social choice problems as follows.

First, we consider the classic problem of *allocations of indivisible goods*, where one seeks to fairly distribute a a set of indivisible goods among a set of heterogeneous agents. Maximizing the utilitarian welfare in this case is well-studied. Using our reduction, the previously mentioned greedy algorithm for agents with additive utilities, allows us to achieve a leximin optimal lottery over the allocations in polynomial time. For submodular utilities, approximating leximin to a factor better than $(1 - \frac{1}{e})$ is NP-hard. However, by applying our reduction, existing approximation algorithms for utilitarian welfare can be leveraged to prove that a 0.5-approximation can be obtained deterministically, while the best-possible $(1 - \frac{1}{e})$-approximation can be obtained with high probability.

Second, we consider the problem of *giveaway lotteries* (Arbiv and Aumann 2022), where there is an event with limited capacity and groups wish to attend, but only-if they can all be admitted together. Maximizing the utilitarian welfare in this setting can be seen as a knapsack problem, for which there is a well-known FPTAS (fully polynomial-time approximation scheme). Using our reduction, we obtain an

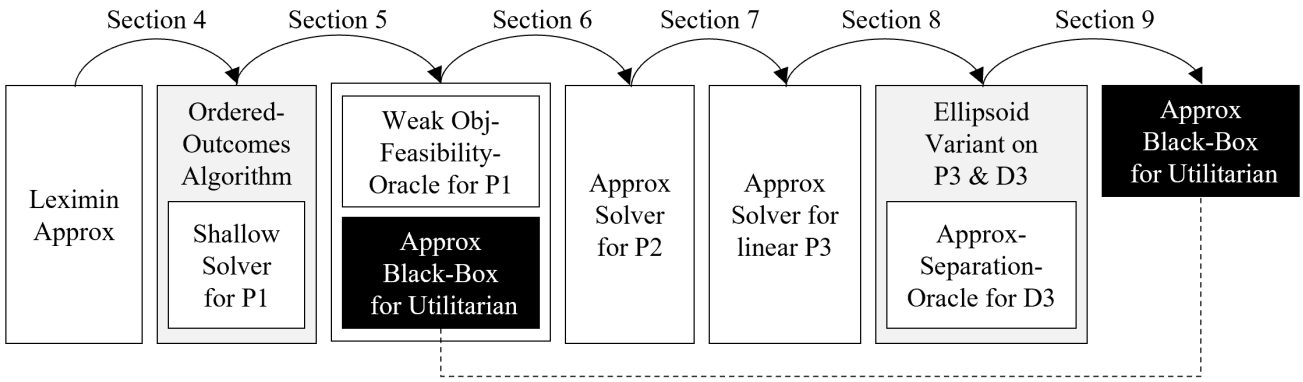

Figure 1: High level description of the reduction algorithm. An arrow from element A to B denotes that the corresponding section reduces problem A to B. White components are implemented in this paper; gray components represent existing algorithms; the black component is the black-box for the utilitarian welfare.

FPTAS for leximin as well.

Lastly, we consider the problem of *fair lotteries for participatory budgeting*, where one seeks to find a fair lottery over the possible budget allocations. When agents have additive utilities, maximizing the utilitarian welfare can also be interpreted as a knapsack problem (albeit in a slightly different way), which allows us to provide an FPTAS for leximin.

Organization. Section 3 introduces the model and required definitions.

In Sections 4-10, we prove our main result: an algorithm for finding a leximin-approximation using an approximate black-box for utilitarian welfare, while importantly, preserving the same approximation factor. The reduction is done step by step. Section 4 reduces the problem of leximin-approximation to another problem; then, each Section $k \in \{5, 6, 7, 8, 9\}$ reduces the problem introduced in Section $k-1$ to another problem, where in Section 9 the reduced problem is approximate utilitarian optimization. Section 10 ties the knots to prove the entire reduction, and extends the result to randomized solvers. A schematic description of the reduction structure is provided in Figure 1.

The applications are shown in Section 11. Lastly, Section 12 concludes with some future work directions. Most proofs are deferred to the appendix.

1.1 Related Work

Recently, there has been a wealth of research focused on finding a fair leximin lottery for specific problems. Examples include algorithms proposed for Representative Cohort Selection (Henzinger et al. 2022), Giveaway Lotteries (Arbiv and Aumann 2022), Allocating Unused Classrooms (Kurokawa, Procaccia, and Shah 2018), and Selecting Citizens' Assemblies (Flanigan et al. 2021). This paper, in contrast, provides a general protocol that can be applied to a wide range of problems.

Alongside these, many papers describe general algorithms for exact leximin optimization (Ogryczak 1997; Ogryczak, Pióro, and Tomaszewski 2004; Ogryczak and Śliwiński 2006). These algorithms usually rely on a solver for single-objective problem, which, in our context, is NP-hard. Recently, Hartman et al. (2023) adapted one of these algorithms to work with approximately-optimal solver. However, designing such a solver remains quite challenging. Our work generalizes these approaches by proving that this algorithm still functions with even a weaker type of solver, which we show can be implemented for many problems.

Another significant area of research focuses on leximin approximations. Since leximin fairness involves multiple objectives simultaneously, it is not straightforward to define what leximin approximation is. Several definitions have been proposed. The definition we employ is related to that of (Kleinberg, Rabani, and Tardos 2001; Abernethy, Schapire, and Syed 2024). Other definitions can be found in (Hartman et al. 2023; Henzinger et al. 2022). An extensive comparison of the different definitions, including examples, is provided in Appendix H.

The work closest to ours is (Kawase and Sumita 2020), which laid the foundation for this research. Their paper studies the problem of stochastic allocation of indivisible goods (see Section 11.1 for more details), and proposes a reduction from egalitarian welfare to utilitarian welfare for this specific problem. We extend their work in two ways. First, we extend their approach from the allocation problem to any problem where lotteries make sense. Second, we show that a black-box for the utilitarian welfare is even more powerful, as it can also be used for leximin, rather than the egalitarian welfare.

2 Preliminaries

We denote the set $\{1, \ldots, n\}$ by $[n]$ for $n \in \mathbb{N}$.

Mathematical Programming. Throughout the paper, we frequently use mathematical programming to define optimization problems. A program is characterized by the following three elements. (1) Constraints: used to define the set of feasible solutions, which forms a subset of $\mathbb{R}^m$ for some $m \in \mathbb{N}$. (2) Type: the program can be either a maximization or minimization program. (3) Objective function: assigns an

objective value to each feasible solution. The goal is to find a feasible solution with an optimal objective value (either maximal or minimal, depending on the problem type).

Notations. For a given program, P, the *feasible region*, denoted by $F(\mathrm{P})$, represents the set of vectors that satisfy all the constraints of P. We say that a vector $\mathbf{v} \in F(\mathrm{P})$ is *a feasible solution for P* (interchangeably: a solution for P or feasible for P), and denote its objective value according to the objective function of P by $obj(\mathrm{P}, \mathbf{v})$.

3 Model and Definitions

The setting postulates a set of n agents $N = \{1, \ldots, n\}$, and a set of deterministic options S — this set represents the possible deterministic allocations - in the fair division setting, or the possible budget allocations - in a budgeting application. For simplicity from now on, we refer to S as *states* and number them $S = \{s_1, \ldots, s_{|S|}\}$.

We seek solutions that are *distributions* over states. Formally, an *S-distribution* is a probability distribution over the set S, and X is the set of all such distributions:

$$X = \{\mathbf{x} = (x_1, \ldots, x_{|S|}) \in \mathbb{R}^{|S|}_{\geq 0} \mid \sum_{j=1}^{|S|} x_j = 1\}$$

Importantly, we allow the number of states, $|S|$, to be exponential in n. This implies that even describing a solution requires exponential time. However, our algorithms will return *sparse solutions* – that is, the number of states with positive probability will be polynomial in n. These solutions can be efficiently described in polynomial time – as a list of states with positive probability, along with their corresponding probabilities.

Definition 3.1 (A Poly-sparse Vector)**.** *A vector, $\mathbf{v} \in \mathbb{R}^m_{\geq 0}$ for some $m \in \mathbb{N}$, is a* poly-sparse *if no more than polynomial number (in n) of its entries are non-zero.*

Degenerate-State. We assume that there exists a single degenerate-state, $s_d \in S$, that gives all the agents utility 0. This will allow us to also consider sub-probabilities over the other states, which give positive utility to some agents. The use of degenerate-state makes sense in our setting, as we assume all utilities are non-negative.

Utilities. The utility of agent $i \in N$ from a given state is described by the function $u_i \colon S \to \mathbb{R}_{\geq 0}$, which is provided as a value oracle.[1]

The utility of agent i from a given S-distribution, $\mathbf{x} \in X$, is the expectations $E_i(\mathbf{x}) = \sum_{j=1}^{|S|} x_j \cdot u_i(s_j)$. The vector of expected utilities of all agents from a solution $\mathbf{x}$ is denoted by $\mathbf{E}(\mathbf{x}) = (E_1(\mathbf{x}), \ldots, E_n(\mathbf{x}))$; and referred to as the expected vector of $\mathbf{x}$.

[1]This means that the algorithm does not have a direct access to the utility function. Rather, given $s \in S$, the value $u_i(s)$ can be obtained in $\mathcal{O}(1)$ for any $i = 1, \ldots, n$.

3.1 Leximin Fairness

We aim for a leximin-optimal S-distribution: one that maximizes the smallest expected-utility, then, subject to that, maximizes the second-smallest expected-utility, and so on. It is formally defined below.

The Leximin Order. For $\mathbf{v} \in \mathbb{R}^n$, let $\mathbf{v}^{\uparrow}$ be the corresponding vector sorted in non-decreasing order, and $v_i^{\uparrow}$ the i-th smallest element (counting repetitions). For example, if $\mathbf{v} = (1, 4, 7, 1)$ then $\mathbf{v}^{\uparrow} = (1, 1, 4, 7)$, and $v_3^{\uparrow} = 4$.

For $\mathbf{v}, \mathbf{u} \in \mathbb{R}^n$, we say that $\mathbf{v}$ is (weakly) *leximin-preferred* over $\mathbf{u}$, denoted $\mathbf{v} \succeq \mathbf{u}$, if one of the following holds. Either $\mathbf{v}^{\uparrow} = \mathbf{u}^{\uparrow}$ (in which case they are *leximin-equivalent*, denoted $\mathbf{v} \equiv \mathbf{u}$). Or there exists an integer $1 \leq k \leq n$ such that $v_i^{\uparrow} = u_i^{\uparrow}$ for $i < k$, and $v_k^{\uparrow} > u_k^{\uparrow}$ (in which case $\mathbf{v}$ is *strictly* leximin-preferred over $\mathbf{u}$, denoted $\mathbf{v} \succ \mathbf{u}$).

Note that the $\mathbf{E}(\mathbf{x})$'s are n-tuples, so the leximin order applies to them.

Observation 3.1. *Let $\mathbf{v}, \mathbf{u} \in \mathbb{R}^n$. Exactly one of the following holds: either $\mathbf{v} \succeq \mathbf{u}$ or $\mathbf{u} \succ \mathbf{v}$.*

Leximin Optimal. We say that $\mathbf{x}^* \in X$ is a *leximin-optimal S-distribution* if $\mathbf{E}(\mathbf{x}^*) \succeq \mathbf{E}(\mathbf{x})$ for all $\mathbf{x} \in X$.

Approximation. Throughout the paper, we denote by $\alpha \in (0, 1]$ a multiplicative approximation ratio.

Definition 3.2 (Leximin-Approximation)**.** *We say that an S-distribution, $\mathbf{x}^A \in X$, is an α-*leximin-approximation *if $\mathbf{E}(\mathbf{x}^A) \succeq \alpha \cdot \mathbf{E}(\mathbf{x})$ for all $\mathbf{x} \in X$.*

Observation 3.2. *An S-distribution is a 1-leximin-approximation if-and-only-if it is leximin-optimal.*

3.2 Utilitarian Optimization

Utilitarian Optimal. We say that $\mathbf{x}^{uo} \in X$ is a *utilitarian-optimal S-distribution* if it maximizes the sum of expected utilities:

$$\forall \mathbf{x} \in X \colon \quad \sum_{i=1}^{n} E_i(\mathbf{x}^{uo}) \geq \sum_{i=1}^{n} E_i(\mathbf{x})$$

Stochasticity is Unnecessary. In fact, for utilitarian welfare, there always exists a deterministic solution – a single state – that maximizes the sum of expected utilities.

Lemma 3.3. *Let $s^{uo} \in \arg\max_{s \in S} \sum_{i=1}^{n} u_i(s)$. Then*

$$\forall \mathbf{x} \in X \colon \quad \sum_{i=1}^{n} u_i(s^{uo}) \geq \sum_{i=1}^{n} E_i(\mathbf{x}).$$

Algorithms for utilitarian welfare are typically designed for this deterministic setting (where the goal is to find a utilitarian-optimal *state*). Our reduction requires a deterministic solver.

Utilitarian Solver. The proposed reduction requires a utilitarian welfare solver that is robust to scaling each utility function by a different constant. Formally:

(I) α-Approximate Black-Box for Utilitarian Welfare

Input: n non-negative constants $c_1, \ldots, c_n$.

Output: A state, $s^{uo} \in S$, for which:

$$\forall s \in S: \quad \sum_{i=1}^{n} c_i \cdot u_i(s^{uo}) \geq \alpha \sum_{i=1}^{n} c_i \cdot u_i(s).$$

When $\alpha = 1$, we say that we have an *exact* black-box.

Many existing solvers for the utilitarian welfare are inherently robust to scaling the utility functions by a constant, as they handle a class of utilities that are closed under this operation. For instance, in the division of goods, various classes of utilities, such as additive and submodular, are closed under constant scaling.

At times, however, rescaling can result in diverging from the problem definition. For example, if the problem definition assumes that utilities are normalized to sum up to 1, then the definition is not robust to rescaling. In such cases, technically, a polynomial solver need not be able to solve the scaled version. Hence, we explicitly include the assumption that the solver is robust to rescaling.

4 Main Loop

The main algorithm used in the reduction is described in Algorithm 1. It is adapted from the Ordered Outcomes algorithm of (Ogryczak and Śliwiński 2006) for finding an exact leximin-optimal solution. It uses a given solver for the maximization program below.

The Program P1. The program is parameterized by an integer $t \in N$ and $t-1$ constants $(z_1, \ldots, z_{t-1})$; its only variable is $\mathbf{x}$ (a vector of size $|S|$ representing an S-distribution):

$$\max \quad \sum_{i=1}^{t} E_i^{\uparrow}(\mathbf{x}) - \sum_{i=1}^{t-1} z_i \qquad \text{(P1)}$$

$$s.t. \quad \text{(P1.1)} \quad \sum_{j=1}^{|S|} x_j = 1$$

$$\text{(P1.2)} \quad x_j \geq 0 \qquad j = 1, \ldots, |S|$$

$$\text{(P1.3)} \quad \sum_{i=1}^{\ell} E_i^{\uparrow}(\mathbf{x}) \geq \sum_{i=1}^{\ell} z_i \qquad \forall \ell \in [t-1]$$

Constraints (P1.1–2) simply ensure that $\mathbf{x} \in X$. Constraint (P1.3) says that for any $\ell < t$, the sum of the smallest ℓ expected-utilities is at least the sum of the ℓ constants $z_1, \ldots, z_\ell$. The objective of a solution[2] $\mathbf{x} \in F(\text{P1})$ is the difference between the sum of its smallest t expected-utilities and the sum of the $t-1$ constants $z_1, \ldots, z_{t-1}$.

[2]See Preliminaries for more details.

Algorithm 1: Main Loop

Input: A solver for P1.

1: **for** $t = 1$ to n **do**
2: Let $\mathbf{x}^t$ be the solution returned by applying the given solver with parameters t and $(z_1, \ldots, z_{t-1})$.
3: Let $z_t := obj(\text{P1}, \mathbf{x}^t)$.
4: **end for**
5: **return** $\mathbf{x}^n$.

Algorithm 1. The algorithm has n iterations. In each iteration t, it uses the given solver for P1 with the following parameters: the iteration counter t, and $(z_1, \ldots, z_{t-1})$ that were computed in previous iterations. The solver returns a solution for this P1, denoted by $\mathbf{x}^t$. If $t < n$ then its objective value, denoted by z_t, is used in the following iterations as an additional parameter. Finally, the solution $\mathbf{x}^n$ generated at the last application of the solver is returned by the main loop.

Notice that the program evolves in each iteration. For any $t \in [n-1]$, the program at iteration $t+1$ differs from the program at iteration t in two ways: first, the objective function changes; and second, an additional constraint is introduced as part of Constraint (P1.3), $\sum_{i=1}^{t} E_i^{\uparrow}(\mathbf{x}) \geq \sum_{i=1}^{t} z_i$ (for $\ell = t$). This constraint is equivalent to: $\sum_{i=1}^{t} E_i^{\uparrow}(\mathbf{x}) - \sum_{i=1}^{t-1} z_i \geq z_t$, which essentially ensures that any solution for following programs achieves an objective value at-least z_t according to the objective function of the program at iteration t. In other words, this constraint guarantees that as we continually improving the situation. This implies, in particular, that $\mathbf{x}^t$ remains feasible for the $(t+1)$-th program.

Observation 4.1. *Let $t \in [n-1]$. The solution obtained in the t-th iteration of Algorithm 1, $\mathbf{x}^t$, is also feasible for the $(t+1)$-th program.*

Solving P1 Exactly. By Ogryczak and Śliwiński (2006) (Theorem 1), the returned $\mathbf{x}^n$ is leximin-optimal when the solver for P1 is *exact* - that is, it returns a solution $\mathbf{x}^t \in F(\text{P1})$ with optimal objective value — $obj(\text{P1}, \mathbf{x}^t) \geq obj(\text{P1}, \mathbf{x})$ for any solution $\mathbf{x} \in F(\text{P1})$.

However, in some cases, no efficient exact solver for P1 is known. An example is stochastic allocation of indivisible goods among agents with submodular utilities, described in Section 11.1 — in this case, it is NP-hard even for $t = 1$, as its optimal solution maximizes the egalitarian welfare (see (Kawase and Sumita 2020) for the hardness proof).

Solving P1 Approximately. Our initial attempt to deal with this issue was to follow the approach of Hartman et al. (2023). They consider an *approximately-optimal* solver for P1, which returns a solution $\mathbf{x}^t \in F(\text{P1})$ with approximately-optimal objective value — $obj(\text{P1}, \mathbf{x}^t) \geq \alpha \cdot obj(\text{P1}, \mathbf{x})$ for any solution $\mathbf{x} \in F(\text{P1})$.

When $t = 1$, the algorithm of Kawase and Sumita (2020) is an approximate-solver for P1. However, for $t > 1$, their technique no longer works due to fundamental differences in the structure of the resulting programs. Designing such a

solver is very challenging; all our efforts to design such a solver for several NP-hard problems were unsuccessful.

Solving P1 Shallowly. Our first contribution is to show that Algorithm 1 can work with an even weaker kind of solver for P1, that we call a *shallow solver*. The term "shallow" is used since the solver returns a solution whose objective value is optimal only with respect to a *subset* of the feasible solutions, as described below.

Recall that s_d is the degenerate-state that gives utility 0 to all agents, and x_d is its probability according to $\mathbf{x}$. We consider the set of solutions that use at-most α fraction of the distribution for the other states, which give positive utility to some agents:

$$X_{\leq\alpha} = \{\mathbf{x} \in X \mid \sum_{j \neq d} x_j \leq \alpha\}.$$

A *shallow-solver* is defined as follows:

(II) α-Shallow-Solver for P1

Input: An integer $t \in N$ and rationals $z_1, \ldots, z_{t-1}$.

Output: A solution $\mathbf{x}^t \in F(\text{P1})$ such that $obj(\text{P1}, \mathbf{x}^t) \geq obj(\text{P1}, \mathbf{x})$ for any $\mathbf{x} \in F(\text{P1}) \cap X_{\leq\alpha}$.

In words: the solver returns a solution $\mathbf{x}^t \in F(\text{P1})$ whose objective value is guaranteed to be optimal comparing only to solutions that are also in $X_{\leq\alpha}$. This is in contrast to an exact solver, where the objective value of the returned solution is optimal comparing to all solutions.[3] Clearly, when $\alpha = 1$, we get an exact solver, as $X_{\leq 1} = X$.

Notice that $\mathbf{x}^t$ does not required to be in $X_{\leq\alpha}$, so its objective value might be *strictly-higher* than the optimal objective value of the set $F(\text{P1}) \cap X_{\leq\alpha}$.

Lemma 4.2. *Given an α-shallow-solver for P1, Algorithm 1 returns an α-leximin-approximation.*

Proof sketch. Consider the solution $\mathbf{x}^n$ returned by Algorithm 1. As $\mathbf{x}^n$ is feasible for the program solved in the last iteration, and as constraints are only being added along the way, we can conclude that $\mathbf{x}^n$ is feasible for all of the n programs solved during the algorithm run.

Next, we suppose by contradiction that $\mathbf{x}^n$ is *not* an α-leximin-approximation. By definition, this means that there exists an $\mathbf{x} \in X$ such that $\alpha\mathbf{E}(\mathbf{x}) \succ \mathbf{E}(\mathbf{x}^n)$. That is, there exists an integer $k \in [n]$ such that: $\alpha E_i^{\uparrow}(\mathbf{x}) = E_i^{\uparrow}(\mathbf{x}^n)$ for $i < k$ and $\alpha E_k^{\uparrow}(\mathbf{x}) > E_k^{\uparrow}(\mathbf{x}^n)$.

We then construct $\mathbf{x}'$ from $\mathbf{x}$ as follows: $x'_j := \alpha \cdot x_j$ for any $j \neq d$, and $x'_d = 1 - \alpha \sum_{j \neq d} x_j$. It is easy to verify that $\mathbf{x}' \in X_{\leq\alpha}$ and that $E_i(\mathbf{x}') = \alpha E_i(\mathbf{x})$ for any i. This means that $E_i^{\uparrow}(\mathbf{x}') = E_i^{\uparrow}(\mathbf{x}^n)$ for $i < k$ (as $\alpha E_i^{\uparrow}(\mathbf{x}) = E_i^{\uparrow}(\mathbf{x}^n)$).

Next, we consider the program solved in iteration $t = k$. As $\mathbf{x}^n$ satisfies all its constraints, and as these constraints impose a lower bound on the sum of the least $\ell < k$ expected-utilities, we can conclude that $\mathbf{x}'$ satisfies all these constraints as well. Thus, $\mathbf{x}' \in F(\text{P1}) \cap X_{\leq\alpha}$.

[3] See Table 1 in Appendix B for comparison of the solvers.

Finally, we prove that the objective-value of $\mathbf{x}'$ for this program is strictly-higher than the one obtained by the shallow-solver (i.e., z_k) — in contradiction to the solver guarantees. □

5 A Shallow-Solver for P1

Our next task is to design a shallow solver for P1. We use a *weak objective-feasibility-oracle*, defined as follows:

(III) α-Weak Objective-Feasibility-Oracle for P1

Input: An integer $t \in N$ and rationals $z_1, \ldots, z_{t-1}$, and another rational z_t.

Output: One of the following claims regarding z_t:

Feasible	$\exists \mathbf{x} \in F(\text{P1})$ s.t. $obj(\text{P1}, \mathbf{x}) \geq z_t$. In this case, the oracle returns such $\mathbf{x}$.
Infeasible Under-$X_{\leq\alpha}$	$\nexists \mathbf{x} \in F(\text{P1}) \cap X_{\leq\alpha}$ s.t. $obj(\text{P1}, \mathbf{x}) \geq z_t$.

Note that these claims are not mutually exclusive, as z_t can satisfy both conditions simultaneously. In this case, the oracle may return any one of these claims.

Lemma 5.1. *Given an α-weak objective-feasibility-oracle for P1 (III), an α-approximate black-box for the utilitarian welfare (I), and an arbitrary vector in $F(P1)$. An efficient α-shallow-solver for P1 (II) can be designed.*

Proof sketch. The solver is described in Algorithm 2 in Appendix C. We perform binary search over the potential objective-values z_t for the program P1.

As a lower bound for the search, we simply use the objective value of the given feasible solution.

As an upper bound for the search, we use an upper bound on the utilitarian welfare of the original utilities u_i – it is sufficient as the objective function is the sum of the t-smallest utilities minus a positive constant; thus, an upper bound on the sum of all utilities can bound the maximum objective value as well. We obtain this upper bound by using the given α-approximate black-box with $c_i = 1$ for all $i \in N$; and then take $\frac{1}{\alpha}$ times the returned value.

During the binary search, we query the weak objective-feasibility-oracle about the current value of z_t. If the oracle asserts Feasible, we increase the lower bound; otherwise, we decrease the upper bound.

We stop the binary search once we reach the desired level of accuracy.[4] Finally, we return the solution $\mathbf{x}$ corresponding to the highest z_t for which the oracle returned Feasible.

By the definition of the oracle, this $\mathbf{x}$ is feasible for P1. In addition, this z_t is at-least as high as the optimal objective across $X_{\leq\alpha}$ – as all the values higher than z_t that were considered by the algorithm, were determined as Infeasible-Under-$X_{\leq\alpha}$. This gives us a shallow solver for P1. □

[4] For simplicity, we assume the binary search error is negligible, as it can be reduced below ϵ in time $\mathcal{O}(\log \frac{1}{\epsilon})$, for any $\epsilon > 0$; the full proof in Appendix C, omits this assumption.

6 Weak Objective-Feasibility-Oracle for P1

To design a weak objective-feasibility-oracle for P1, we modify P1 as follows. First, we convert the optimization program P1, to a feasibility program (without an objective), by adding a constraint saying that the objective function of P1 is at least the given constant z_t: $\sum_{i=1}^{t} E_i^{\uparrow}(\mathbf{x}) \geq \sum_{i=1}^{t} z_i$. We then make two changes to this feasibility program: (1) remove Constraint (P1.1) that ensures that the sum of values is 1, and (2) add the objective function: $\min \sum_{j=1}^{|S|} x_j$. We call the resulting program P2:

$$\min \quad \sum_{j=1}^{|S|} x_j \quad s.t. \tag{P2}$$

$$\text{(P2.1)} \quad x_j \geq 0 \qquad\qquad j = 1, \ldots, |S|$$

$$\text{(P2.2)} \quad \sum_{i=1}^{\ell} E_i^{\uparrow}(\mathbf{x}) \geq \sum_{i=1}^{\ell} z_i \qquad \forall \ell \in [t]$$

Note that (P2.2) contains the constraints (P1.3), as well as the new constraint added when converting to a feasibility program.

As before, the only variable is $\mathbf{x}$. However, in this program, a vector $\mathbf{x}$ can be feasible without being in X, as its elements are not required to sum to 1.

We shall now prove that a weak objective-feasibility-oracle for P1 can be designed given a solver for P2, which returns a poly-sparse[5] approximately-optimal solution. As P2 is a *minimization* program and as $\alpha \in (0, 1]$, it is defined as follows:

(IV) $\frac{1}{\alpha}$-Approx.-Optimal-Sparse-Solver for P2

Input: An integer $t \in N$ and rationals $z_1, \ldots, z_t$.

Output: A poly-sparse solution $\mathbf{x}^A \in F(\text{P2})$ such that $obj(\text{P2}, \mathbf{x}^A) \leq \frac{1}{\alpha} obj(\text{P2}, \mathbf{x})$ for any $\mathbf{x} \in F(\text{P2})$.

Lemma 6.1. *Given an $\frac{1}{\alpha}$-approximately-optimal-sparse-solver for P2 (IV), an α-weak objective-feasibility-oracle for P1 (III) can be obtained.*

Proof. We apply the solver to get a vector $\mathbf{x}^A \in F(\text{P2})$, and then check whether $obj(\text{P2}, \mathbf{x}^A) := \sum_{j=1}^{|S|} x_j^A$ is at-most 1. Importantly, this can be done in polynomial time, since $\mathbf{x}^A$ is a poly-sparse vector.

If so, we assert that z_t is Feasible. Indeed, while $\mathbf{x}^A$ may not be feasible for P1, we can construct another poly-sparse vector, from $\mathbf{x}^A$, that is feasible for P1 and achieves an objective value of at-least z_t. The new vector, $\mathbf{x}'$, is equal to $\mathbf{x}^A$ except that $x'_d := (1 - \sum_{j=1}^{|S|} x_j^A)$. It can be easily verified that the new vector, $\mathbf{x}'$, is in $F(\text{P1})$, and that $obj(\text{P1}, \mathbf{x}') \geq z_t$, as required.

[5] A poly-sparse vector (def. 3.1) is one whose number of non-zero values can be bounded by a polynomial in n.

Otherwise, $\sum_{j=1}^{|S|} x_j^A > 1$. In this case, we assert that z_t is Infeasible-under-$X_{\leq\alpha}$. Indeed, assume for contradiction that there exists $\mathbf{x} \in F(\text{P1}) \cap X_{\leq\alpha}$ with $obj(\text{P1}, \mathbf{x}) \geq z_t$. We show that this implies that the optimal (min.) objective-value for (P2) is at-most α, meaning that any $\frac{1}{\alpha}$-approximation should yield an objective value of at-most 1 – in contradiction to the objective value of $\mathbf{x}^A$. We construct a new vector $\mathbf{x}'$ from $\mathbf{x}$, where $\mathbf{x}'$ equals to $\mathbf{x}$ except that $x'_d := 0$. As $\mathbf{x} \in F(\text{P1})$ and $E_i(\mathbf{x}) = E_i(\mathbf{x}')$ for all $i \in N$, it follows that $\mathbf{x}' \in F(\text{P2})$. But $\mathbf{x} \in X_{\leq\alpha}$, which means that it uses at-most α for states $j \neq d$. This implies that $obj(\text{P2}, \mathbf{x}') := \sum_{j=1}^{|S|} x'_j$ is at-most α, which ensures that the optimal (min.) objective is also at-most α – as required. □

7 Approximately-Optimal Solver for P2

The use of $E^{\uparrow}()$ operator makes both P1 and P2 non-linear. However, Ogryczak and Śliwiński (2006) showed that P1 can be "linearized" by replacing the constraints using $E^{\uparrow}()$ with a polynomial number of linear constraints. We take a similar approach for P2 to construct the following *linear* program P3:

$$\min \quad \sum_{j=1}^{|S|} x_j \quad s.t. \tag{P3}$$

$$\text{(P3.1)} \quad x_j \geq 0 \qquad\qquad j = 1, \ldots, |S|$$

$$\text{(P3.2)} \quad \ell y_\ell - \sum_{i=1}^{n} m_{\ell,i} \geq \sum_{i=1}^{\ell} z_i \qquad \forall \ell \in [t]$$

$$\text{(P3.3)} \quad m_{\ell,i} \geq y_\ell - \sum_{j=1}^{|S|} x_j \cdot u_i(s_j) \qquad \forall \ell \in [t],\ \forall i \in [n]$$

$$\text{(P3.4)} \quad m_{\ell,i} \geq 0 \qquad\qquad \forall \ell \in [t],\ \forall i \in [n]$$

Constraints (P3.2–4) introduce $t(n+1) \leq n(n+1)$ auxiliary variables: y_ℓ and $m_{\ell,i}$ for all $\ell \in [t]$ and $i \in [n]$. We formally prove the equivalence between the two sets of constraints in Appendix D; and show that it implies that the required solver for P2 can be easily derived from the same type of solver for P3.

(V) $\frac{1}{\alpha}$-Approx.-Optimal-Sparse-Solver for P3

Input: An integer $t \in N$ and rationals $z_1, \ldots, z_t$.

Output: A poly-sparse $\left(\mathbf{x}^A, \mathbf{y}^A, \mathbf{m}^A\right) \in F(\text{P3})$ such that $obj(\text{P3}, \left(\mathbf{x}^A, \mathbf{y}^A, \mathbf{m}^A\right)) \leq \frac{1}{\alpha} obj(\text{P3}, (\mathbf{x}, \mathbf{y}, \mathbf{m}))$ for any $(\mathbf{x}, \mathbf{y}, \mathbf{m}) \in F(\text{P3})$.

Lemma 7.1. *Given an $\frac{1}{\alpha}$-approximately-optimal-sparse-solver for P3 (V), an $\frac{1}{\alpha}$-approximately-optimal-sparse-solver for P2 (IV) can be obtained.*

Proof sketch. The equivalence between the two sets of constraints says that $(\mathbf{x}, \mathbf{y}, \mathbf{m}) \in F(\text{P3})$ if-and-only-if $\mathbf{x} \in F(\text{P2})$. Thus, given a poly-sparse $\frac{1}{\alpha}$-approximately-optimal

solution $\left(\mathbf{x}^A, \mathbf{y}^A, \mathbf{m}^A\right) \in F(\text{P3})$; then, $\mathbf{x}^A$ (which is clearly a poly-sparse vector), is a $\frac{1}{\alpha}$-approximately-optimal solution for P2. □

8 Approximately-Optimal Solver for P3

P3 is a linear program, but has more than $|S|$ variables. Although $|S|$ (the number of states) may be exponential in n, P3 can be approximated in polynomial time using a variant of the ellipsoid method, similarly to Karmarkar and Karp (1982), as described bellow. The method uses an approximate separation oracle for the dual of the linear program P3:

$$\max \quad \sum_{\ell=1}^{t} q_\ell \sum_{i=1}^{\ell} z_i \quad s.t. \tag{D3}$$

$$\text{(D3.1)} \quad \sum_{i=1}^{n} u_i(s_j) \sum_{\ell=1}^{t} v_{\ell,i} \leq 1 \qquad \forall j = 1, \ldots, |S|$$

$$\text{(D3.2)} \quad \ell q_\ell - \sum_{i=1}^{n} v_{\ell,i} \leq 0 \qquad \forall \ell \in [t]$$

$$\text{(D3.3)} \quad -q_\ell + v_{\ell,i} \leq 0 \qquad \forall \ell \in [t], \ \forall i \in [n]$$

$$\text{(D3.4)} \quad q_\ell \geq 0 \qquad \forall \ell \in [t]$$

$$\text{(D3.5)} \quad v_{\ell,i} \geq 0 \qquad \forall \ell \in [t], \ \forall i \in [n]$$

Similarly to P3, the program D3 is parameterized by an integer t, and rational numbers $(z_1, \ldots, z_t)$. It has a polynomial number of variables: q_ℓ and $v_{\ell,j}$ for any $\ell \in [t]$ and $j \in [n]$; and a potentially exponential number of constraints due to (D3.1); see Appendix E for derivation.

We prove that the required solver for P3 can be designed given the following procedure for its dual D3:

(VI) $\frac{1}{\alpha}$-Approx.-Separation-Oracle for D3

Input: An integer $t \in N$, rationals $z_1, \ldots, z_t$, and a potential assignment of the program variables $(\mathbf{q}, \mathbf{v})$.

Output: One of the following regarding $(\mathbf{q}, \mathbf{v})$:

Infeasible	At least one of the constraints is violated by $(\mathbf{q}, \mathbf{v})$. In this case, the oracle returns such a constraint.
$\frac{1}{\alpha}$-Approx. Feasible	All the constraints are $\frac{1}{\alpha}$-approximately-maintained — the left-hand side of the inequality is at least $\frac{1}{\alpha}$ times the its right-hand side.

In Appendix F, we present the variant of the ellipsoid method that, given a $\frac{1}{\alpha}$-approximate-separation oracle for the (max.) dual program, allows us to obtain a sparse $\frac{1}{\alpha}$-approximation to the (min.) primal program (Lemma F.1). This allows us to conclude the following:

Corollary 8.1. *Given a $\frac{1}{\alpha}$-approximate-separation-oracle for D3 (VI), a $\frac{1}{\alpha}$-approximately-optimal-sparse-solver for P3 (V) can be derived.*

9 Approx. Separation Oracle for D3

Now, we design the required oracle using the given approximate black-box for the utilitarian welfare.

Lemma 9.1. *Given an α-approximate black-box for the utilitarian welfare (I), a $\frac{1}{\alpha}$-approximate-separation-oracle for D3 (VI) can be constructed.*

Proof. The oracle can be designed as follows. Given an assignment of the variables of the program D3, Constraints (D3.2–5) can be verified directly, as their number is polynomial in n. If a violated constraint was found, the oracle returns it. Otherwise, the potentially exponential number of constraints in (D3.1) are treated collectively. We operate the approximate black-box with $c_i := \sum_{\ell=1}^{t} v_{\ell,i}$ for $i \in N$, and obtain a state $s_k \in S$. If $\sum_{i=1}^{n} c_i \cdot u_i(s_k) > 1$, we declare that constraint k in (D3.1) is violated. Otherwise, we assert that the assignment is approximately-feasible. Indeed, in this case, Constraints (D3.2)-(D3.5) are exactly-maintained (since we first check them directly), and all the constraints in (D3.1) are approximately-maintained, as proven below. By the definition of the approximate black-box, we can conclude that $\sum_{i=1}^{n} c_i \cdot u_i(s_k) \geq \alpha \sum_i c_i \cdot u_i(s)$ for any state $s \in S$. It follows that, for any state $s \in S$, the corresponding constraint in (D3.1) is approximately-maintained:

$$\sum_{i=1}^{n} u_i(s) \sum_{\ell=1}^{t} v_{\ell,i} \leq \frac{1}{\alpha} \sum_{i=1}^{n} u_i(s_k) \sum_{\ell=1}^{t} v_{\ell,i} \leq \frac{1}{\alpha} \cdot 1$$

□

10 The Main Result

Putting it all together, we obtain:

Theorem 10.1. *Given an α-approximate black-box for the utilitarian welfare (I). An α-leximin-approximation (Def. 3.2) can be computed in time polynomial in n and the running time of the black-box.*

Proof. By Lemma 9.1, given an α-approximate black-box for utilitarian welfare, a $\frac{1}{\alpha}$-approximate-separation-oracle for D3 can be constructed.

By Corollary 8.1, using this oracle, we can construct a $\frac{1}{\alpha}$-approximately-optimal-sparse-solver for P3.

By Lemma 7.1, we can use this solver to design a $\frac{1}{\alpha}$-approximately-optimal-sparse-solver for P2.

By Lemma 6.1, this solver allows us to construct an α-weak objective-feasibility-oracle for P1.

By Lemma 5.1, a binary search with (1) this procedure, (2) the given α-approximate black-box for utilitarian welfare, and (3) an arbitrary solution for P1 (as follows); allows us to design an α-shallow-solver for P1. As an arbitrary solution, for $t = 1$, we take $\mathbf{x}^0$ defined as the S-distribution with $x^0_d = 1$ and $x^0_j = 0$ for $j \neq d$. However, for $t \geq 2$, we rely on the fact that the shallow solver is used within the iterative Algorithm 1, and take the solution returned in the previous iteration, $\mathbf{x}^{t-1}$, which, by Observation 4.1, is feasible for the program at iteration t as well.

By Lemma 4.2, when this shallow solver is used inside Algorithm 1, the output is an α-leximin approximation. □

Together with Observation 3.2, this implies:

Corollary 10.2. *Given an exact black-box for the utilitarian welfare, a leximin-optimal S-distribution can be obtained in polynomial time.*

10.1 Randomized Solvers

A *randomized* black-box returns a state that α-approximates the utilitarian welfare with probability $p > 0$, otherwise returns an arbitrary state. Theorem 10.1 can be extended as follows:

Theorem 10.3. *Given a randomized α-approximate black-box for the utilitarian welfare with success probability $p \in (0, 1)$. An α-leximin-approximation can be obtained with the same success probability p in time polynomial in n and the running time of the black-box.*

Proof sketch. We first prove that the use of the randomized black-box does not effect feasibility — that is, the output returned by Algorithm 1 is always an S-distribution. Then, we prove the required guarantees regarding its optimally.

Recall that the black-box is used only in two places – to obtain an upper bound for the binary search over the potential objective-values for P1 (Section 5), and as part of the separation oracle for D3 (Section 9).

Inside the binary search, if the obtained value does not approximate the optimal utilitarian welfare, it might cause us to overlook larger possible objective values. While this could affect optimality, it will not impact feasibility.

As for the separation oracle, this change might cause us to determine that Constraint (D3.1) is approximately-feasible even though it is not. However, in Appendix F.1, we prove that even with this change, the solution produced by the ellipsoid method remains feasible for the primal (although its optimality guarantees might no longer hold). Consequently, the solution ultimately returned by Algorithm 1 is also feasible (i.e., an S-distribution in X).

We can now move on to the optimally guarantees. Assume that the black-box for the utilitarian welfare is randomized and has a success probability $p < 1$. It is clear that iteratively calling this solver within the algorithm reduces the overall success probability.

To address this, we first boost the success probability of each iteration by calling the original black-box multiple times (on the same instance) and taking the best result among these calls. This boosts the success probability of the iteration (as failure corresponds to the probability that none of the calls to the original black-box was successful). We prove that with an appropriate choice for the number of such repetitive calls, the total success probability of the entire algorithm can be as high as that of the original black-box (p), while maintaining polynomial complexity. □

11 Applications

This section provides three applications of our general reduction framework. Each application employs a different black-box for the associated utilitarian welfare.

11.1 Stochastic Indivisible Allocations

In the problem of *fair stochastic allocations of indivisible goods*, described by Kawase and Sumita (2020), there is a set of m indivisible goods, G, that needs to be distributed fairly among the n agents.

The Set S. The states are the possible allocations of the goods to the agents. Each state can be described by a function mapping each good to the agent who gets it. Accordingly, $S = \{s \mid s\colon G \to N\}$, and $|S| = n^m$.

We assume that agents care only about their own share, so we can abuse notation and let each u_i take a bundle B of goods. The utilities are assumed to be normalized such that $u_i(\emptyset) = 0$, and monotone – $u_i(B_1) \leq u_i(B_2)$ if $B_1 \subseteq B_2$. Under these assumptions, different black-boxes for the utilitarian welfare exist.

The Utilitarian Welfare. For any n constants $c_1, \ldots, c_n$, the goal is to maximize the following:

$$\max_{s \in S} \sum_{i=1}^{n} c_i \cdot u_i(s)$$

Many algorithms for approximating utilitarian welfare are already designed for classes of utilities, which are closed under multiplication by a constant. This means that, given such an algorithm for the original utilities $(u_i)_{i \in N}$, we can use it as-is for the utilities $(c_i \cdot u_i)_{i \in N}$.

Results. When the utilities are additive, maximizing the utilitarian welfare can be done in polynomial time by greedily giving each item to the agent who values it the most. This means, according to Corollary 10.2, that:

Corollary 11.1. *For additive utilities, a leximin-optimal S-distribution can be obtained in polynomial time.*

When the utilities are submodular, approximating leximin to a factor better than $(1 - \frac{1}{e})$ is NP-hard (Kawase and Sumita 2020).[6] However, as there is a deterministic $\frac{1}{2}$-approximation algorithm for the utilitarian welfare (Fisher, Nemhauser, and Wolsey 1978), by Theorem 10.1:

Corollary 11.2. *For submodular utilities, a $\frac{1}{2}$-leximin-approximation can be found in polynomial time.*

There is also a randomized $(1 - \frac{1}{e})$-approximation algorithm for the case where utilities are submodular, with high success probability (Vondrak 2008). Thus, by Theorem 10.3:

Corollary 11.3. *For submodular utilities, a $(1-\frac{1}{e})$-leximin-approximation can be obtained with high probability in polynomial time.*

11.2 Giveaway Lotteries

In *giveaway lotteries*, described by Arbiv and Aumann (2022), there is an event with a limited capacity, and groups who wish to attend it - but only-if all the members of the

[6] Kawase and Sumita (2020) prove that approximating the egalitarian welfare to a factor better than $(1 - \frac{1}{e})$ is NP-hard. However, since an α-leximin-approximation is first-and-foremost an α-approximation to the egalitarian welfare, the same hardness result applies to leximin as well.

group can attend together. Here, each group of people is an agent. We denote the size of group i by $w_i \in \mathbb{N}_{\geq 0}$ and the event capacity by $W \in \mathbb{N}_{\geq 0}$. It is assumed that $w_i \leq W$ for $i \in N$ and $\sum_{i \in N} w_i > W$.[7]

The Set S. Each state describes a set of the groups that can attend the event together: $S = \{s \subseteq N \mid \sum_{i \in s} w_i \leq W\}$. Here, $|S|$ is only bounded by 2^n.

The utility of group $i \in N$ from a state s is 1 if they being chosen according to s (i.e., if $i \in s$) and 0 otherwise.

The Utilitarian Welfare. For any n constants $c_1, \ldots, c_n$, the goal is to maximize the following:

$$\max_{s \in S} \sum_{i=1}^{n} c_i \cdot u_i(s) = \max_{s \in S} \sum_{i \in s} c_i$$

This is just a knapsack problem with n item (one for each group), where the weights are the group sizes w_i (as we only look at the legal packing $s \in S$), and the values are the constants c_i.

Result. It is well known that there is an FPTAS for the Knapsack problem. By Theorem 10.1:

Corollary 11.4. *There exists an FPTAS for leximin for the problem of giveaway lotteries.*

11.3 Participatory Budgeting Lotteries

The problem of *fair lotteries for participatory budgeting*, was described by Aziz et al. (2024). Here, the n agents are voters, who share a common budget $B \in \mathbb{R}_{>0}$ and must decide which projects from a set P to fund. Each voter, $i \in N$, has an *additive* utility over the set of projects, u_i; while the projects have costs described by $cost\colon P \to \mathbb{R}_{>0}$. [8]

The set S. The states are the subsets of projects that fit in the given budget: $S = \{s \subseteq P \mid cost(s) \leq B\}$. The size of S in this problem is only bounded by $2^{|P|}$.

The Utilitarian Welfare. For any n constants $c_1, \ldots, c_n$, the goal is to maximize the following:

$$\max_{s \in S} \sum_{i=1}^{n} c_i \cdot u_i(s) = \max_{s \in S} \sum_{i=1}^{n} \sum_{p \in s} c_i \cdot u_i(p) \quad \text{(Additivity)}$$

$$= \max_{s \in S} \sum_{p \in s} \left(\sum_{i=1}^{n} c_i \cdot u_i(p) \right)$$

This can also be seen as a knapsack problem where: the items are the projects, the weights are the costs, and the value of item $p \in P$ is $\sum_{i=1}^{n} c_i \cdot u_i(p)$.

Result. As before, the existence of a FPTAS for the Knapsack problem together with Theorem 10.1, give:

Corollary 11.5. *There is an FPTAS for leximin for participatory budgeting lotteries.*

[7] Arbiv and Aumann (2022) provide an algorithm to compute a leximin-optimal solution. However, their algorithm is polynomial only for a unary representation of the capacity.

[8] Aziz et al. (2024) study fairness properties based on *fair share* and *justified representation*.

12 Conclusion and Future Work

In this work, we establish a strong connection between leximin fairness and utilitarian optimization, demonstrated by a reduction. It is robust to errors in the sense that, given a black-box that approximates the utilitarian value, a leximin-approximation with respect to the same approximation factor can be obtained in polynomial time.

Negative Utilities. Our current technique requires to assume that the utilities are non-negative since we use the degenerate-state. It remains unclear whether a similar reduction can be obtained for non-positive utilities. We note that combining positive and negative utilities poses even more challenges – as even the meaning of multiplicative approximation in this case is unclear.

Nash Welfare. The Nash welfare (product of utilities) is another prominent objective in social choice, offering a compelling compromise between the efficiency of utilitarianism and the fairness of egalitarianism. From a computational perspective, maximizing Nash welfare is typically as challenging as maximizing egalitarian welfare. An interesting question is whether a similar reduction can be constructed from Nash welfare (in expectation) to utilitarian optimization.

Applications. We believe this method can also be applied to a variety of other problems, such as: Selecting a Representative Committee (Henzinger et al. 2022), Allocating Unused Classrooms (Kurokawa, Procaccia, and Shah 2018), Selecting Citizens' Assemblies (Flanigan et al. 2021), Cake-Cutting (Aumann, Dombb, and Hassidim 2013; Elkind, Segal-Halevi, and Suksompong 2021), Nucleolus (Elkind et al. 2009).

Best-of-both-worlds. Aziz et al. (2024), who study participatory budgeting lotteries, focus on fairness that is both ex-ante (which is a guarantee on the distribution) and ex-post (which is a guarantee on the deterministic support). In this paper, we guarantee only ex-ante fairness. Can ex-post fairness be achieved alongside it? We note that our method ensures both ex-ante and ex-post efficiency – since leximin guarantees Pareto-optimality with respect to the expected utilities E_i, any state in the support is Pareto-optimal with respect to the utilities u_i.

Deterministic Setting. When the objective is to find a leximin-optimal (or approximate) *state* rather than a distribution, it remains an open question whether a black-box for utilitarian welfare can still contribute, even if for a different approximation factor.

Truthfulness. Arbiv and Aumann (2022), who study giveaway lotteries, prove that a leximin-optimal solution is not only fair but also truthful in this case. Can our leximin-approximation be connected to some notion of truthfulness?

Leximin-Approximation Definitions. This paper suggests a weaker definition of leximin-approximation than the one proposed by Hartman et al. (2023) (see Appendix H for more details). Can similar results be obtained with the stronger definition?

Acknowledgments

This research is partly supported by the Israel Science Foundation grants 712/20, 1092/24, 2697/22, and 3007/24. We sincerely appreciate Arkadi Nemirovski, Yasushi Kawase, Nikhil Bansal, Shaddin Dughmi, Edith Elkind, and Dinesh Kumar Baghel for their valuable insights, helpful answers, and clarifications. We are also grateful to the following members of the stack exchange network for their very helpful answers to our technical questions: Neal Young, 1Rock, Mark L. Stone, Rob Pratt, mtanneau, and mhdadk. Lastly, we would like to thank the reviewers in AAAI 2025 for their helpful comments.

References

Abernethy, J. D.; Schapire, R.; and Syed, U. 2024. Lexicographic optimization: Algorithms and stability. In *International Conference on Artificial Intelligence and Statistics*, 2503–2511. PMLR.

Arbiv, T.; and Aumann, Y. 2022. Fair and Truthful Giveaway Lotteries. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5): 4785–4792.

Aumann, Y.; Dombb, Y.; and Hassidim, A. 2013. Computing socially-efficient cake divisions. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, 343–350. IFAAMAS.

Aziz, H.; Lu, X.; Suzuki, M.; Vollen, J.; and Walsh, T. 2024. Fair Lotteries for Participatory Budgeting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 9469–9476.

Bansal, N.; and Sviridenko, M. 2006. The Santa Claus problem. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, 31–40. New York, NY, USA: Association for Computing Machinery. ISBN 1595931341.

Bei, X.; Lu, X.; and Suksompong, W. 2022. Truthful Cake Sharing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5): 4809–4817.

Cheng, H.; Kong, S.; Deng, Y.; Liu, C.; Wu, X.; An, B.; and Wang, C. 2023. Exploring Leximin Principle for Fair Core-Selecting Combinatorial Auctions: Payment Rule Design and Implementation. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, 2581–2588. ijcai.org.

Elkind, E.; Goldberg, L. A.; Goldberg, P. W.; and Wooldridge, M. J. 2009. On the computational complexity of weighted voting games. *Ann. Math. Artif. Intell.*, 56(2): 109–131.

Elkind, E.; Segal-Halevi, E.; and Suksompong, W. 2021. Mind the Gap: Cake Cutting With Separation. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, 5330–5338. AAAI Press.

Fisher, M. L.; Nemhauser, G. L.; and Wolsey, L. A. 1978. *An analysis of approximations for maximizing submodular set functions—II*, 73–87. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-00790-3.

Flanigan, B.; Gölz, P.; Gupta, A.; Hennig, B.; and Procaccia, A. D. 2021. Fair algorithms for selecting citizens' assemblies. *Nature*, 596(7873): 548–552.

Flanigan, B.; Liang, J.; Procaccia, A. D.; and Wang, S. 2024. Manipulation-Robust Selection of Citizens' Assemblies. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Proceedings of the AAAI Conference on Artificial Intelligence*, 9696–9703. AAAI Press.

Freeman, R.; Sikdar, S.; Vaish, R.; and Xia, L. 2019. Equitable Allocations of Indivisible Goods. In Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-2019*, 280–286. ijcai.org.

Hartman, E.; Hassidim, A.; Aumann, Y.; and Segal-Halevi, E. 2023. Leximin Approximation: From Single-Objective to Multi-Objective. In *ECAI 2023*, 996–1003. IOS Press.

Henzinger, M.; Peale, C.; Reingold, O.; and Shen, J. H. 2022. Leximax Approximations and Representative Cohort Selection. In Celis, L. E., ed., *3rd Symposium on Foundations of Responsible Computing (FORC 2022)*, volume 218 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 2:1–2:22. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-226-6.

Karmarkar, N.; and Karp, R. M. 1982. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 312–320. Chicago, IL, USA: IEEE.

Kawase, Y.; and Sumita, H. 2020. On the Max-Min Fair Stochastic Allocation of Indivisible Goods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02): 2070–2078.

Kleinberg, J.; Rabani, Y.; and Tardos, E. 2001. Fairness in Routing and Load Balancing. *Journal of Computer and System Sciences*, 63(1): 2–20.

Kurokawa, D.; Procaccia, A. D.; and Shah, N. 2018. Leximin Allocations in the Real World. *ACM Trans. Economics and Comput.*, 6(3-4): 11:1–11:24.

Ogryczak, W. 1997. On the lexicographic minimax approach to location problems. *European Journal of Operational Research*, 100(3): 566–585.

Ogryczak, W.; Pióro, M.; and Tomaszewski, A. 2004. Telecommunications network design and max-min optimization problem. *Journal of telecommunications and information technology*, 4: 43–53.

Ogryczak, W.; and Śliwiński, T. 2006. On Direct Methods for Lexicographic Min-Max Optimization. In Gavrilova, M.; Gervasi, O.; Kumar, V.; Tan, C. J. K.; Taniar, D.; Laganá, A.; Mun, Y.; and Choo, H., eds., *Computational Science and Its Applications - ICCSA 2006*, 802–811. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-34076-8.

Vondrak, J. 2008. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 67–74. Victoria British Columbia Canada: ACM. ISBN 978-1-60558-047-0.