Fairness in Temporal Slot Assignment

Edith Elkind, Sonja Kraiczy, and Nicholas Teh

Abstract

We investigate settings where projects need to be assigned to time slots based on preferences of multiple agents. We consider a variety of objectives, including utilitarian social welfare, egalitarian social welfare, Nash social welfare, Pareto optimality, equitability, and proportionality. We introduce a general-purpose randomized algorithm, which, for each of these objectives, can decide whether it is achievable for a given instance; the running time of this algorithm is in the complexity class XP with respect to the number of agents. We also provide complexity results for the case where the number of agents is large, and identify special cases that admit efficient algorithms.

1 Introduction

> "Where there is an observatory and a telescope, we expect that any eyes will see new worlds at once."
>
> —*Henry David Thoreau (1817–1862)*

Thoreau's words were recorded over a century ago, and yet they continue to ring true today. In July 2022, NASA's James Webb Space Telescope recorded breathtaking images of the distant cosmic cliffs and the glittering landscape of star birth in fine detail. The stunning images were made possible by the advancement of space exploring technologies, prompting widespread awe both within the scientific community and the world at large. Unsurprisingly, it is clear that no regular institution, let alone individual, could afford such highly specialized equipment for their own (research) endeavours. The new discoveries were funded by a 10 billion-dollar investment from NASA and the USA.

NASA, whose goal is to "expand knowledge for the benefit of humanity", could choose to loan its facilities (not just this telescope, but others as well) to (possibly independent) research institutes to further the exploration of space, maybe with fresh perspectives. In fact, on a smaller scale, in 2008, libraries in the US trialed the idea for loaning out telescopes, pioneered by the New Hampshire Astronomical Society. The movement has proved to be a great success, and currently, over 100 libraries across the country participates in this program. The city of Westminster in the UK also has a similar program.

Now, each institution may have their preferred schedule of equipment rental—some of them may want to see certain specific galaxies or phenomena, and due to natural conditions, they can only view them during certain time periods [24]. Then, the goal of NASA is to come up with a schedule for the loan of their observatories and telescopes. In doing so, multiple goals may be considered. For instance, if maximal utilization is the primary goal, then maximizing *utilitarian* welfare may be desirable, to fully utilize the telescope by filling up its schedule as much as possible. However, NASA may wish to give different institutions the opportunity to gain access to the facility and hopefully bring in new perspectives, and thus, may wish to treat different institutions fairly. Thus, other welfare objectives, such as maximizing *egalitarian* welfare, or fairness notions such as *equitability* and *proportionality* may become more relevant instead.

Other scenarios where different agents' preferences over schedules (i.e., assignments of activities to time slots) need to be aggregated into a common schedule include scheduling

university lectures, conference talks or popular entertainment. Inspired by these applications, in this work, we study the problem faced by a central authority in creating a common schedule in a fair and efficient manner. In particular, we consider the computational problems associated with achieving outcomes that satisfy various welfare and fairness desiderata.

1.1 Our Contributions

We introduce a model that enables us to study welfare and fairness properties that arise in the assignment of projects to time slots when agents have approval preferences over pairs of the form (project, time slot). Throughout the paper, we assume that each project is assigned to exactly one time slot, and each time slot implements exactly one project.

To begin, we focus on the case of constant number of agents, and obtain randomized polynomial-time algorithms for all the fairness and welfare notions discussed. Also, for two agents, we show how to find an outcome that is proportional up to one slot. We then investigate the case of arbitrary number of agents. We present an efficient method for obtaining outcomes that maximize utilitarian welfare, and NP-hardness results for maximizing egalitarian welfare and for finding outcomes that are equitable or proportional. For egalitarian welfare, we obtain positive algorithmic results for an interesting special case of our problem.

1.2 Related Work

The concept of allocating projects to time slots is related to long-term (or perpetual) participatory budgeting (PB) [20, 21]; however, the focus of PB is primarily on the budgetary process, rather than on distributive fairness, affecting the definitions of fairness considered.

Some works in the social choice literature that consider the temporal element in distributive justice include [14, 15], but these works primarily focus on the dynamic setting (i.e., online reporting of preferences), and specifically on how one may obtain fair outcomes in an online fashion. Other works that look into the off-line case (albeit in different settings and considering different notions of fairness) include [4] and works in dynamic resource allocation [23].

Other works in the scheduling [17, 18] and matching literature look into various applications such as house allocation [32], project matching [1], and job scheduling [33], amongst others. However, these models are often more complex, taking into account contextual variables, making the study of welfare and fairness notions we consider difficult.

The temporal slot assignment problem can also be re-interpreted in a spatial context, as assigning projects to plots of land (see, e.g., the work of [10]).

2 Preliminaries

We first present our basic model, and then introduce the fairness notions that we will discuss.

2.1 Model

We are given a set $N = \{1, \ldots, n\}$ of n *agents*, a set $P = \{p_1, \ldots, p_m\}$ of m distinct *projects*, and a set $T = \{t_1, \ldots, t_\ell\}$ of ℓ time slots, where $\ell \leq m$. For each time slot $r \in [\ell]$, each agent $i \in N$ specifies a subset of projects $s_{i,r} \subseteq P$: these are the projects that i approves to be implemented at time r. We write $\mathbf{s}_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,\ell})$, and refer to the list $\mathbf{s}_i$ as i's *approval set*. For notational simplicity, if $s_{i,r}$ is a singleton, we sometimes omit the set notation. Equivalently, we can represent agents' preferences as graphs: for each agent $i \in N$ her *approval graph* G^i is a bipartite graph with parts P and T that contains an edge

$(p_j, t_r) \in P \times T$ if and only if $p_j \in s_{i,r}$. We let $G^i_{j,r} = 1$ if $(p_j, t_r) \in G^i$, and $G^i_{j,r} = 0$ otherwise. Let μ_i be the size of a maximum matching in G^i.

An *outcome* $\mathbf{o} = (o_1, \ldots, o_\ell)$ is a list of ℓ projects such that for every $r \in [\ell]$ the project $o_r \in P$ is chosen to be built at time slot t_r. We require that for any $r, r' \in [\ell]$ with $r \neq r'$, we have $o_r \neq o_{r'}$; that is, each project is built at most once. Let Π denote the space of all possible outcomes.

The utility of an agent $i \in N$ from an outcome $\mathbf{o}$ is the number of matches between the agent's preference and the outcome: $u_i(\mathbf{o}) = |\{r \in [\ell] : o_r \in s_{i,r}\}|$. Note that an outcome maximizes agent i's utility if and only if it corresponds to a maximum matching in G^i, i.e., provides her a utility of μ_i.

Preference restrictions In general, our framework allows for agents to approve of any number of projects between 0 and m for each slot. From this, we can derive three natural restricted cases of our model: (1) limiting the number of approved projects for *each* slot, (2) limiting the *total* number of approved projects across all slots, and (3) limiting the *number of times* a project can be approved overall. The special case where each agent approves exactly one project for each slot and each project is approved exactly once by each agent is called the *full permutation* (FP) setting. The setting where for each $i \in N$ each project appears in at most one set $s_{i,r}$ and $|\cup_{r\in[\ell]} s_{i,r}| \leq k$ for some $\gamma \in [m]$ is called the *γ-partial permutation* (γ-PP) setting.

2.2 Welfare and Fairness Concepts

The goal of this work is to study the algorithmic complexity of identifying good outcomes in our model. There are several ways to define what it means for an outcome to be good. In what follows, we formally define the notions of goodness that will be considered in the remainder of the paper.

Perhaps the most straightforward approach is to focus on outcomes that optimize some notion of welfare.

Definition 2.1 (Utilitarian Social Welfare)**.** An outcome $\mathbf{o}$ *maximizes utilitarian social welfare (Max-UTIL)* if for every outcome $\mathbf{o}' \in \Pi$ it holds that $\sum_{i\in N} u_i(\mathbf{o}) \geq \sum_{i\in N} u_i(\mathbf{o}')$.

Definition 2.2 (Egalitarian Social Welfare)**.** An outcome $\mathbf{o}$ *maximizes egalitarian social welfare (Max-EGAL)* if for every outcome $\mathbf{o}' \in \Pi$ it holds that $\min_{i\in N} u_i(\mathbf{o}) \geq \min_{i\in N} u_i(\mathbf{o}')$.

Definition 2.3 (Nash Social Welfare)**.** An outcome $\mathbf{o}$ *maximizes Nash social welfare (Max-NASH)* if for every outcome $\mathbf{o}' \in \Pi$ it holds that $\prod_{i\in N} u_i(\mathbf{o}) \geq \prod_{i\in N} u_i(\mathbf{o}')$.

Another relevant notion of welfare is Pareto optimality.

Definition 2.4 (Pareto Optimality)**.** An outcome $\mathbf{o}$ is *Pareto optimal (PAR)* if for any outcome $\mathbf{o}' \in \Pi$ either $u_i(\mathbf{o}) \geq u_i(\mathbf{o}')$ for all $i \in N$, or there exists an agent $i \in N$ with $u_i(\mathbf{o}) > u_i(\mathbf{o}')$.

Instead of maximizing the welfare, we may want to focus on equity: can we obtain an outcome that guarantees each agent the same utility? To capture this idea, we define the following notion of fairness.

Definition 2.5 (Equitability)**.** An outcome $\mathbf{o}$ is *equitable (EQ)* if for all $i, i' \in N$ it holds that $u_i(\mathbf{o}) = u_{i'}(\mathbf{o})$.

Note that an equitable outcome does not always exist (this is also the case for many similar fairness properties in the social choice literature [6]). Consider the simple case of two agents, with approval sets $\mathbf{s}_1 = (p_1, p_2)$ and $\mathbf{s}_2 = (p_2, p_1)$ respectively. Then, any outcome $\mathbf{o}$ will give a utility of 2 to one agent, and 0 to the other—equal treatment of these individuals is not achievable in our model.

We will also consider another fairness property that is commonly studied in the context of fair division, namely, proportionality. Intuitively, this property demands that each agent's utility should be at least as high as her proportional fair share. In our setting, it is natural to define an agent's proportional fair share as her best-case utility $\max_{\mathbf{o}\in\Pi} u_i(\mathbf{o}) = \mu_i$, divided by the number of agents n. Thus, we define proportionality as follows.

Definition 2.6 (Proportionality)**.** An outcome $\mathbf{o}$ is *proportional (PROP)* if for all $i \in N$ it holds that $u_i(\mathbf{o}) \geq \frac{\mu_i}{n}$.

However, the existence of PROP outcomes is not guaranteed, even in the FP setting. Indeed, consider again the two-agent instance with $\mathbf{s}_1 = (p_1, p_2)$ and $\mathbf{s}_2 = (p_2, p_1)$. A PROP outcome would have to provide each agent a utility of 1, but both possible outcomes give a utility of 2 to one agent and 0 to the other. Thus, just like for equitability, we consider an "up to k slots" relaxation.

Definition 2.7 (Proportionality up to k slots)**.** An outcome $\mathbf{o}$ is *proportional up to k slots (PROPk)* if, for all $i \in N$ it holds that $u_i(\mathbf{o}) \geq \frac{M_i}{n} - k$.

In the context of proportionality, we are particularly interested in the case $k = 1$ (i.e., PROP1).

3 Welfare and Fairness with Constant Number of Agents

In this section we present positive algorithmic results for the case where the number of agents is bounded by a constant. Notably, our results hold for the most general variant of our model (i.e., arbitrary preferences).

3.1 A General-Purpose Randomized Algorithm

We present a general-purpose method that gives rise to randomized algorithms capable of solving a variety of problems efficiently (in polynomial time) if the number of agents is bounded by a constant. The problems we consider include deciding if there exists an outcome providing a certain level of utilitarian/egalitarian/Nash social welfare, Pareto optimality, equitability, proportionality, etc. Furthermore, our method can be extended to solve the search variants of these problems efficiently. To simplify presentation, we assume that the number of projects m is equal to the number of time slots ℓ, i.e., outcomes are perfect matchings between projects and time slots; for the case of $\ell < m$, we can simply include additional timeslots with empty approval sets for each agent to make $m = \ell$. The running time of our algorithms is polynomial in m^n, i.e., it is polynomial when n is bounded by a constant.

The key idea of our approach is to summarize the agents' preferences by means of a matrix whose entries are monomials over variables $y_1, \ldots, y_n, y_{n+1}$ (encoding the agents) and $(x_{j,r})_{j,r\in[m]}$ (encoding the project-time slot pairs); then, by evaluating the determinant of this matrix, we can determine if our instance admits a 'good' schedule.

The starting point for our construction is the Edmonds matrix, which is used to reason about matchings in bipartite graphs [26].

Definition 3.1 (Edmonds Matrix [26])**.** Let $(x_{j,r})_{j,r\in[m]}$ be indeterminates. Given a balanced bipartite graph $G = (P \cup T, E)$ with parts $P = \{p_1, \dots p_m\}$ and $T = \{t_1, \dots t_m\}$, the *Edmonds matrix* is a matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ with entries A_{jr} defined as follows:

$$A_{jr} = \begin{cases} x_{j,r} & \text{if } (p_j, t_r) \in E \\ 0 & \text{otherwise} \end{cases}$$

The Edmonds matrix is useful for deciding if a balanced bipartite graph contains a perfect matching [26]: it is not hard to check that its determinant is not identically zero (as a polynomial) if and only if G admits a perfect matching.

However, we cannot use the Edmonds matrix directly: indeed, the input to our problem is not a single bipartite graph, but a collection of n bipartite graphs $G^1, \dots, G^n$. We therefore generalize Edmonds' idea as follows.

Definition 3.2 (Polynomial Schedule Matrix)**.** Let $(y_i)_{i\in[n+1]}$ and $(x_{j,r})_{j,r\in[m]}$ be indeterminates. The *polynomial schedule matrix* for $G^1, \dots, G^n$ is a matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$ with entries A_{jr} defined as follows:

$$A_{jr} = \begin{cases} y_{n+1} \cdot x_{j,r} & \text{if } G^i_{j,r} = 0 \text{ for all } i \in [n] \\ \prod_{i=1}^{n} y_i^{G^i_{j,r}} \cdot x_{j,r} & \text{otherwise} \end{cases}$$

3.1.1 Using the polynomial schedule matrix

We will now describe how to use the polynomial schedule matrix to check for existence of good outcomes. Set $\mathbf{y_N} = (y_i)_{i\in[n+1]}$, $\mathbf{x_E} = (x_{j,r})_{j,r\in[m]}$, and denote the determinant of $\mathbf{A}$ by $det(\mathbf{A}) = f(\mathbf{y_N}, \mathbf{x_E})$. The function f is a polynomial in $\mathbf{y_N}$, $\mathbf{x_E}$. It will also be convenient to think of $det(\mathbf{A})$ as a polynomial in $\mathbf{y_N}$ whose coefficients are polynomials over $\mathbf{x_E}$; when using this interpretation, we will write $det(\mathbf{A}) = g(\mathbf{y_N})$.

The function $f(\mathbf{y_N}, \mathbf{x_E})$ can be written as a sum of $m!$ monomials. Each of these monomials is of the form $(-1)^z \cdot y_1^{v_1} \dots y_n^{v_n} y_{n+1}^{v_{n+1}} \cdot x_{j_1,r_1} \cdot \dots \cdot x_{j_m,r_m}$, where $z \in \{0, 1\}$, $v_1, \dots, v_{n+1}$ are non-negative integers, and $\{j_1, \dots, j_m\} = \{r_1, \dots, r_m\} = [m]$; for each $i \in [n]$ the value v_i is the utility that agent i derives from the schedule that assigns project p_{j_s} to time slot t_{r_s} for $s \in [m]$.

We can rephrase the reasoning above in terms of g. Specifically, we can represent $g(\mathbf{y_N})$ as a sum of at most $(m+1)^{n+1}$ monomials over $\mathbf{y_N}$, with the coefficient of each monomial being a polynomial over $\mathbf{x_E}$. Specifically, each monomial in $g(\mathbf{y_N})$ is an expression of the form $y_1^{v_1} \dots y_n^{v_n} y_{n+1}^{v_{n+1}} \lambda(\mathbf{x_E})$, where $0 \le v_i \le m$ for each $i \in [n+1]$ and $\lambda(\mathbf{x_E})$ is a polynomial over $\mathbf{x_E}$; if λ is not identically zero, there are one or more outcomes providing utility v_i to agent i for $i \in [n]$ and containing v_{n+1} project-time slot pairs not approved by any of the agents.

It follows that as long as we know which monomials occur in g, we can decide whether our instance admits outcomes that satisfy each of the welfare and fairness desiderata defined in Section 2. In more detail[1]:

- **Egalitarian Social Welfare**: Given $v_1, \dots, v_n \ge 0$, we can determine whether there exists an outcome $\mathbf{o}$ such that $u_i(\mathbf{o}) \ge v_i$ for each $i \in [n]$, by checking whether g contains a monomial of the form $\prod_{i=1}^{n+1} y_i^{v'_i} \lambda(\mathbf{x_E})$ such that $v'_i \ge v_i$ for all $i \in [n]$.
- **Nash Social Welfare**: The value of a Nash welfare-maximizing outcome can be computed efficiently by iterating through all monomials in g: given a monomial of the

[1] We omit a discussion of utilitarian social welfare, since an outcome maximizing this measure of welfare can be computed in polynomial time for arbitrary n, m and ℓ (Theorem 4.1).

form $\prod_{i=1}^{n+1} y_i^{v_i}\lambda(\mathbf{x_E})$, we compute $\prod_{i=1}^{n} v_i$, and take the maximum of this quantity over all monomials that appear in g.

- **Pareto Optimality**: For each $i \in [n]$, let $v_i = u_i(\mathbf{o})$. Then if g has a term $\prod_{i=1}^{n+1} y_i^{v_i'}\lambda(\mathbf{x_E})$ with $v_i' \geq v_i$ for all $i \in [n]$ and $v_i' > v_i$ for at least one $i \in [n]$, then $\mathbf{o}$ is not Pareto optimal, otherwise it is.

- **Equitability**: Given $v \geq 0$, we can determine whether there exists an outcome $\mathbf{o}$ such that $u_i(\mathbf{o}) = v$ for each $i \in [n]$, by checking whether there exists a term $\prod_{i=1}^{n+1} y_i^{v}\lambda(\mathbf{x_E})$ such that $v = v_i$ for all $i \in [n]$.

- **Proportionality** can be handled similarly to egalitarian social welfare by setting $v_i = \mu_i/n$ for each $i \in [n]$.

More broadly, having access to g enables us to solve any problem where the answer depends on whether there exists an outcome that provides certain utilities to each of the agents. We will now discuss the complexity of deciding which monomials appear in g.

3.1.2 Evaluating g

We have argued that we can decide the existence of 'good' outcomes by evaluating $\det(\mathbf{A})$ (more specifically, by determining which monomials appear in g). However, explicitly listing all $m!$ terms of $f(\mathbf{y_N}, \mathbf{x_E})$ is computationally expensive. Instead, we will now describe a probabilistic polynomial-time algorithm that with high probability correctly identifies which monomials appear in g. Formally, we will say that a polynomial $\widehat{g}$ over $\mathbf{y_N}$ is an *over-approximation* of g if for every choice of $v_1, \ldots, v_{n+1} \in \{0, \ldots, m\}$ the following holds: if g contains a monomial of the form $\prod_{i=1}^{n+1} y_i^{v_i}\lambda(\mathbf{x_E})$ such that λ is not identically 0, then $\widehat{g}$ contains a monomial of the form $\widehat{\lambda} \cdot \prod_{i=1}^{n+1} y_i^{v_i}$ with $\widehat{\lambda} \in \mathbb{R} \setminus \{0\}$. Also, we say that $\widehat{g}$ is an *under-approximation* of g if for every choice of $v_1, \ldots, v_{n+1} \in \{0, \ldots, m\}$ the following holds: if g does not contain a monomial of the form $\prod_{i=1}^{n+1} y_i^{v_i}\lambda(\mathbf{x_E})$, then $\widehat{g}$ does not contain a monomial of the form $\widehat{\lambda} \cdot \prod_{i=1}^{n+1} y_i^{v_i}$. Our algorithm will output a polynomial $\widehat{g}$ over $\mathbf{y_N}$ such that $\widehat{g}$ is an under-approximation of g, and, with probability at least $1/2$, it is an over-approximation of g. Note that if $\widehat{g}$ is both an under-approximation and an over-approximation of g, it contains at most $(m+1)^{n+1}$ terms, and it can be used instead of g to solve all fair scheduling problems considered in this section. Moreover, the probability that $\widehat{g}$ is an over-approximation of g can be amplified using standard techniques.

To obtain $\widehat{g}$ given the matrix $\mathbf{A}$, we sample values for the variables in $\mathbf{x_E}$ from the set $\lceil dm \rceil$ for $d = 2 \cdot (m+1)^{n+1}$, and substitute them into $\mathbf{A}$, so as to obtain a matrix $\mathbf{A'}$ whose entries are polynomials in $\mathbf{y_N}$. We then set $\widehat{g} = det(\mathbf{A'})$.

Computing the determinant is easy for real-valued matrices: e.g., one can obtain the LU decomposition [31, 7] of the input matrix in time $\mathcal{O}(m^{2.376})$ time using the Coppersmith–Winograd algorithm for matrix multiplication, and then compute the determinant of the simpler matrices in the decomposition in linear-time. However, the entries of $\mathbf{A'}$ are multivariate polynomials rather than reals. We will therefore use polynomial interpolation techniques. That is, we select values $y_1', \ldots, y_{n+1}' \in \mathbb{R}$, substitute them into $\mathbf{A'}$, and compute the determinant of the resulting real-valued matrix using the LU decomposition approach. By repeating this step multiple times (selecting fresh values of $y_1', \ldots, y_{n+1}' \in \mathbb{R}$ at each iteration and computing $\widehat{g}(y_1', \ldots, y_{n+1}')$), we obtain enough information to recover $\widehat{g}$. Specifically, Zippel's deterministic method takes $\mathcal{O}(n^2 m^{n+1} t^2)$ time, where t is the true number of non-zero terms in $\widehat{g}$; whereas Zippel's probabilistic method [35] provides an improved expected running time of $\mathcal{O}(n^2 m t^2)$. An overview of methods for polynomial interpolation and their complexity can be found in [35, 16].

It is clear that $\widehat{g}$ is an under-approximation of g. To show that it is an over-approximation of g with probability at least $1/2$, we use the Schwartz–Zippel lemma [26]:

Proposition 3.3. *Set $d = 2\cdot(m+1)^{n+1}$, and let $S = \lceil dm \rceil$. Then picking $x'_{j,r}$ independently, uniformly at random from S gives a polynomial in $y_1, \ldots, y_{n+1}$ such that the coefficient of every term $\prod_{i=1}^{n+1} y_i^{v_i}$ that appears in $det(\mathbf{A})$ is non-zero with probability at least $1/2$.*

Proof. Fix some values $v_1, \ldots, v_{n+1}$. The coefficient of $\prod_{i=1}^{n+1} y_i^{v_i}$ is a polynomial in $\mathbf{x_E}$ of degree at most m. By the Schwartz–Zippel lemma [26], the probability that this coefficient is zero is at most $\frac{m}{|S|} = \frac{1}{d}$. By the union bound, the probability that this happens for some choice of $v_1, \ldots, v_{n+1}$ is at most $\frac{1}{d} \cdot (m+1)^{n+1} = 1/2$. □

The running time of our algorithm is polynomial in $(m+1)^{n+1}$, i.e., it is polynomial in m if n is bounded by a constant. Indeed, our algorithm proceeds by constructing $\mathbf{A}$, sampling values for $\mathbf{x_E}$, substituting them into $\mathbf{A}$, and evaluating the determinant of the resulting matrix using polynomial intepolation; this determinant is a multivariate polynomial that contains at most $(m+1)^{n+1}$ monomials, and, given this polynomial, the fair scheduling problems we are interested in can be solved in polynomial time.

3.1.3 Extension to search problems

We can extend our approach from decision problems to search problems. To illustrate this, we consider the problem of finding an outcome that maximizes the utilitarian welfare.

Corollary 3.4. *Suppose we are given a non-negative integer v_i for each $i \in [n]$. Then we can efficiently find an outcome $\mathbf{o}$ such that $u_i(\mathbf{o}) \geq v_i$ for all $i \in [n]$, or report that no such outcome exists.*

Sketch. Initialize the solution as $S = \varnothing$. Pick an edge $e \notin S$ from $\cup_{i=1}^{n} G^i$, remove it from each approval graph, and run the randomized polynomial-time algorithm on this problem instance. If a solution still exists, e was not essential, and we may continue removing edges. If there is no solution, we know e must be part of every solution that includes the edges placed in S so far; we reinstate it in the approval graphs, set $S := S \cup \{e\}$, and continue. □

3.2 PROP1 Outcomes Exist for Two Agents

In Section 2, we observed that some instances do not have PROP outcomes, and defined a relaxation of PROP, namely, proportionality up to k items (PROPk). We show that for two agents, PROP1 outcomes always exist and can be computed in polynomial time. Due to space constraints, the proof is deferred to the appendix.

Proposition 3.5. *Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be graphs on the same set of vertices V. Also, for each $i = 1, 2$, let μ_i be the size of a maximum matching in G_i. Then, we can compute, in polynomial time, a matching M in $(V, E_1 \cup E_2)$ that contains at least $\frac{\mu_1}{2} - 1$ edges from E_1 and at least $\frac{\mu_2}{2} - 1$ edges from E_2.*

Corollary 3.6. *When $n = 2$, a* PROP1 *allocation exists and can be computed in polynomial time.*

Proof. Applying Proposition 3.5 to the approval graphs G^1 and G^2 (of agents 1 and 2 respectively) results in a PROP1 allocation.

□

The algorithm described in the proof of Proposition 3.5 runs in polynomial time. More specifically, the most computationally expensive task is computing the maximum matchings in G_1 and G_2, which takes $\mathcal{O}(\sqrt{m} \cdot |E(G_i)|)$ time for $i = 1, 2$. Constructing the graph G', finding the connected components, and defining the valuations takes $\mathcal{O}(\ell)$ time. The adjusted winner procedure is also linear in the number of items, which is at most the number of vertices in G', i.e., at most 2ℓ, so it takes $\mathcal{O}(\ell)$ time. Furthermore, given the divisible component C^*, we can decide if it is a cycle or an alternating path in linear time (by looking at the number of vertices, and the number of 1- and 2- edges each) and hence decide which of its edges to select.

4 Beyond a Constant Number of Agents

In this section, we analyze the complexity of finding good outcomes in scenarios where the number of agents may be large. We obtain an easiness result for utiliarian social welfare, and hardness results for several other solution concepts.

4.1 Utilitarian Social Welfare

The utilitarian social welfare is perhaps the most popular optimization target in multi-agent allocation problems. We start by formalizing the problem of computing an outcome that maximizes this measure of welfare.

> Utilitarian Social Welfare Maximization (Util):
> **Input**: A problem instance $I = (P, T, (\mathbf{s}_i)_{i \in N})$, and a parameter $\lambda \in \mathbb{N}$.
> **Question**: Is there an outcome $\mathbf{o}$ such that $\sum_{i \in N} u_i(\mathbf{o}) \geq \lambda$?

We will now show that the problem we just defined admits an efficient algorithm; in fact, we can even compute an outcome that maximizes the utilitarian social welfare in polynomial time.

Theorem 4.1. Util *is solvable in polynomial time.*

Proof. Given an instance of Util, we construct a weighted complete bipartite graph with parts P and T, where the weight of the edge (p_j, t_r) equals to the number of agents that approve implementing p_j at time t_r: that is, we set $w(p_j, t_r) = |\{i \in N : p_j \in s_{i,r}\}|$. Then a maximum-weight matching in this graph corresponds to an outcome that maximizes the utilitarian social welfare. It remains to observe that a maximum-weight matching in a bipartite graph can be computed in polynomial time [30]. □

4.2 Egalitarian Social Welfare

Next, we consider the complexity of maximizing the egalitarian welfare. Again, we first formulate the associated decision problem.

> Egalitarian Social Welfare Maximization (Egal):
> **Input**: A problem instance $I = (P, T, (\mathbf{s}_i)_{i \in N})$, and a parameter $\lambda \in \mathbb{N}$.
> **Question**: Is there an outcome $\mathbf{o}$ such that $u_i(\mathbf{o}) \geq \lambda$ for each $i \in N$?

It turns out that Egal is NP-complete. In fact, this hardness result holds even in the FP setting.

Our proof proceeds by a reduction from the Binary Closest String Problem (BCSP) [13, 22]. An instance of this problem consists of ν binary strings of length ρ each, and an integer κ; it is a yes-instance if there exists a binary string y of length ρ such that the Hamming distance (i.e., number of *mismatches*) between y and each of the ν strings is at most κ (equivalently, the number of matches is at least $\rho-\kappa$), and a no-instance otherwise. This problem is known to be NP-complete [13, 22].

We are now ready to establish the complexity of Egal.

Theorem 4.2. Egal *is* NP*-complete. The hardness result holds even in the FP setting.*

Proof. It is clear that Egal is in NP: given an outcome, we can evaluate each agent's utility and compare it to λ. To establish NP-hardness, we give a reduction from BCSP.

Consider an instance of BCSP given by ν binary strings $X = \{x_1, \dots, x_\nu\}$ of length ρ each, and an integer κ. For each $i \in [\nu]$, $j \in [\rho]$, denote the j-th bit of the string x_i by x_{ij}. To create an instance of Egal, we introduce 2ρ projects $p_1, \dots, p_{2\rho}$ and 2ρ time slots $t_1, \dots, t_{2\rho}$. We will encode the bit strings as agents' preferences: for each x_{ij}, if $x_{ij} = 1$, let $s_{i,2j-1} = p_{2j-1}$ and $s_{i,2j} = p_{2j}$; if $x_{ij} = 0$, let $s_{i,2j-1} = p_{2j}$ and $s_{i,2j} = p_{2j-1}$.

Let $\lambda = 2(\rho - \kappa)$. We will now prove that there exists an outcome $\mathbf{o}$ that gives each agent a utility of at least λ if and only if there exists a binary string y of length ρ such that its Hamming distance to each string in X is at most κ (i.e., the number of matches is at least $\rho - \kappa$).

For the 'if' direction, let $y = (y_1, \dots, y_\rho)$ be a string with at most κ mismatches to each of the strings in X. Construct an outcome $\mathbf{o}$ by setting $o_{2j-1} = p_{2j-1}$, $o_{2j} = p_{2j}$ if $y_j = 1$, and $o_{2j-1} = p_{2j}$, $o_{2j} = p_{2j-1}$ if $y_j = 0$. Consider an agent i. For each bit j such that $x_{ij} = y_j$ we have $s_{i,2j-1} = o_{2j-1}$, $s_{i,2j} = o_{2j}$. Thus, the utility of this agent is at least $2(\rho - \kappa) = \lambda$, which is what we wanted to prove.

For the 'only if' direction, suppose there exists an outcome that gives each agent a utility of λ. We will say that an outcome $\mathbf{o}$ is *proper* if for each $j \in [\rho]$ we have $\{o_{2j-1}, o_{2j}\} = \{p_{2j-1}, p_{2j}\}$. We claim that there exists a proper outcome that gives each agent a utility of λ. Indeed, suppose that this is not the case. For each outcome $\mathbf{o}$, let $z(\mathbf{o})$ be the number of time slots t_q such that $q \in \{2j-1, 2j\}$ for some $j \in [\rho]$, but $o_q \notin \{p_{2j-1}, p_{2j}\}$. Among all outcomes that give each agent a utility of λ, pick one with the minimal value of $z(\mathbf{o})$; let this outcome be $\mathbf{o}^*$. By our assumption, $\mathbf{o}^*$ is not proper, so $z(\mathbf{o}^*) > 0$. Thus, there exists a time slot t_q such that $q \in \{2j-1, 2j\}$ for some $j \in [\rho]$, but $o^*_q \notin \{p_{2j-1}, p_{2j}\}$. Then in $\mathbf{o}^*$ there is a project $p \in \{p_{2j-1}, p_{2j}\}$ that is scheduled at time slot $t_{2j'-1}$ or $t_{2j'}$ for some $j' \neq j$. Modify $\mathbf{o}^*$ by swapping p with o^*_q. Note that in $\mathbf{o}^*$, no agent derives positive utility from either of these two projects. Hence, this swap cannot decrease any agent's utility, but it decreases $z(\cdot)$ (because the project at time slot t_q is now one of p_{2j-1}, p_{2j}), a contradiction with our choice of $\mathbf{o}^*$.

Now, fix a proper outcome $\mathbf{o}$ that gives each agent a utility of λ. Let

$$y_j = \begin{cases} 1 & \text{if } o_{2j-1} = p_{2j-1} \text{ and } o_{2j} = p_{2j} \\ 0 & \text{if } o_{2j-1} = p_{2j} \text{ and } o_{2j} = p_{2j-1} \end{cases}$$

Consider a string x_i. We know that agent i's utility from $\mathbf{o}$ is at least $\lambda = 2(\rho - \kappa)$. Note that for each $j \in [\rho]$ we have either (1) $o_{2j-1} = s_{i,2j-1}$, $o_{2j} = s_{i,2j}$ or (2) $o_{2j-1} \neq s_{i,2j-1}$, $o_{2j} \neq s_{i,2j}$. Hence, there can be at most κ indices $j \in [\rho]$ for which condition (2) holds, and therefore there are at most κ mismatches between x_i and y. □

Theorem 4.2 indicates that even for FP instances, it is hard to decide whether each agent can be guaranteed a utility of at least λ. This motivates us to consider a less ambitious goal: can Egal be solved efficiently if λ is bounded by a small constant?

Perhaps surprisingly, even for $\lambda = 1$ and FP instances this is unlikely to be the case: we show that Egal is as hard as the Perfect Complete Bipartite Proper Rainbow Matching (PCBP-RM) problem [27], one of the many variants of the Rainbow Matching problem for which an NP-hardness result is conjectured, but has not been established [2].

An instance of PCBP-RM is given by a complete bipartite graph $K_{\nu,\nu}$ (i.e., there are ν nodes on each side of the graph, and each node on one side is connected to every other node on the opposite side), where edges are properly colored (i.e., if two edges share an endpoint, they have different colors). The goal is to decide whether this instance admits a perfect rainbow matching M, i.e., a matching of size ν in which every edge has a different color.

Theorem 4.3. Egal *is at least as hard as* PCBP-RM*, even when restricted to FP instances with* $\lambda = 1$*.*

Proof. Consider an instance of PCBP-RM with two parts V_1 and V_2, $|V_1| = |V_2| = \nu$, where for all $i, j \in [\nu]$, the i-th vertex in V_1 is connected to the j-th vertex in V_2 via an edge e_{ij}; we denote the color of this edge by $\texttt{color}(e_{ij})$. There are a total of ν unique colors $C = \{c_1, \dots, c_\nu\}$.

We construct an instance of Egal that contains ν agents, ν projects, and ν time slots. For each agent i and time slot t_r, we set $s_{i,r} = p_j$, where j is the index of the color of the edge e_{ir}, i.e., $\texttt{color}(e_{ir}) = c_j$.

We will now prove that there exists an outcome $\mathbf{o}$ that gives each agent a utility of at least 1 if and only if there exists a perfect rainbow matching M.

For the 'if' direction, let M be a perfect rainbow matching. We create an outcome $\mathbf{o}$ as follows. To determine the time slot for project p_j, we identify an edge of M that has color c_j; if this edge connects agent i and time slot t_r, we schedule p_j at time t_r (thereby providing utility 1 to agent i). Since M is a rainbow matching, each project is scheduled exactly once, and any two projects are assigned distinct time slots. Moreover, as M is a matching, each agent's utility is 1.

For the 'only if' direction, consider an outcome $\mathbf{o}$ that gives each agent a utility of at least 1. Observe that for each $r \in [\nu]$ and every pair of agents i, i', we have $s_{i,r} \neq s_{i',r}$. This means that for each $r \in [\nu]$, there is exactly one agent that receives a utility of 1 from o_r, i.e., the utility of each agent is exactly 1. We construct the matching M as follows: if agent i receives utility from the project scheduled at t_r, we add an edge from the i-th vertex of V_1 to the r-th vertex of V_2 to M. Note that M is a matching: each vertex in V_1 is matched, as each agent receives utiity 1 from some project, and each vertex in V_2 is matched, as each time slot provides positive utility to at most one agent. Moreover, M is a rainbow matching, as each project is scheduled exactly once. □

On a more positive note, for $\lambda = 1$ in the γ-PP setting we can characterize the complexity of Egal with respect to the parameter γ. We do so by establishing a tight relationship between this problem and the k-SAT problem. An instance of k-SAT consists of ν Boolean variables and ρ clauses, where each clause has at most k literals; it is a yes-instance if there exists an assignment of Boolean values to the variables such that at least one literal in each clause evaluates to True, and a no-instance otherwise. This problem is NP-complete for each $k \geq 3$, but polynomial-time solvable for $k = 2$.

Theorem 4.4. Egal *is* NP*-complete even when restricted to* γ*-PP instances with* $\lambda = 1$*, for any fixed* $\gamma \geq 3$*.*[2]

Proof. We describe a reduction from γ-SAT to Egal restricted to γ-PP instances with $\lambda = 1$.

[2]It is important to note that this does not mean that when $\lambda = 1$, Egal under FP is always NP-complete. We cannot use the $\gamma = m$ argument here, even if $m \geq 3$.

Consider an instance of γ-SAT given by ν Boolean variables $X = \{x_1, \ldots, x_\nu\}$ and ρ clauses $\mathcal{C} = \{C_1, \ldots, C_\rho\}$. In our instance of EGAL, we have a set of ρ agents $N = \{1, \ldots, \rho\}$, 2ν projects $P = \{p_1, \ldots, p_{2\nu}\}$, and 2ν time slots $T = \{t_1, \ldots, t_{2\nu}\}$. Each agent encodes a clause: for each $i \in N$, $j \in [\nu]$ we set

$$s_{i,2j-1} = \begin{cases} p_j & \text{if } C_i \text{ contains the positive literal } x_j \\ \varnothing & \text{otherwise} \end{cases}$$

and

$$s_{i,2j} = \begin{cases} p_j & \text{if } C_i \text{ contains the negative literal } \neg x_j \\ \varnothing & \text{otherwise} \end{cases}$$

As we start with an instance of γ-SAT, we have $|\cup_{r\in[2\nu]} s_{i,r}| \leq \gamma$ for each $i \in N$, i.e., we obtain a valid γ-PP instance.

We will now prove that there exists an outcome $\mathbf{o}$ that gives each agent a positive utility if and only if our instance of γ-SAT admits a satisfying assignment.

For the 'if' direction, consider a satisfying assignment $(\xi_j)_{j\in[\nu]}$. For $j \in [\nu]$, if ξ_j is set to True, let $o_{2j-1} = p_j$, $o_{2j} = p_{\nu+j}$ and if ξ_j is set to False, let $o_{2j-1} = p_{\nu+j}$, $o_{2j} = p_j$. Consider an agent $i \in [\rho]$. Since the assignment $(\xi_j)_{j\in[\nu]}$ satisfies C_i, there is a literal $\ell \in C_i$ that is satisfied by this assignment. If $\ell = x_j$ is a positive literal then ξ_j is set to True, so $o_{2j-1} = p_j$, and we have $s_{i,2j-1} = p_j$. If $\ell = \neg x_j$ is a negative literal then ξ_j is set to False, so $o_{2j} = p_j$, and we have $s_{i,2j} = p_j$. In either case, the utility of agent i is at least 1.

For the 'only if' direction, consider an outcome $\mathbf{o}$ that gives each agent a positive utility. Arguing as in the proof of Theorem 4.2, we can assume that for each $j \in [\nu]$ it holds that p_j is scheduled at t_{2j-1} or at t_{2j}: if this is not the case for some $j \in [\nu]$, we can move p_j to one of these slots without lowering the utility of any agent. We construct a truth assignment $(\xi_j)_{j\in[\nu]}$ by setting ξ_j to True if $o_{2j-1} = p_j$ and to False if $o_{2j} = p_j$. Now, consider a clause C_i. Since the utility of agent i is at least 1, it follows that our assignment satisfies at least one of the literals in C_i. As this holds for all clauses, the proof is complete. □

Theorem 4.5. EGAL *is polynomial-time solvable when restricted to 2-PP instances with* $\lambda = 1$.

Proof. Consider an instance of EGAL with $\lambda = 1$ given by n agents $N = \{1, \ldots, n\}$, m projects $P = \{p_1, \ldots, p_m\}$ and ℓ time slots $T = \{t_1, \ldots, t_\ell\}$. For each project $p_j \in P$ and time slot $t_r \in T$, create a variable x_{jr}; intuitively, we want this variable to evaluate to True if project p_j is scheduled at time slot t_r and to False otherwise.

First, we encode the fact that each project can be scheduled at most once: for each project $p_j \in P$ and each pair of time slots $t_r, t_{r'} \in T$ with $r \neq r'$ we add the clause $\neg x_{jr} \vee \neg x_{jr'}$. Let the conjunction of these clauses be C^*.

Next, we encode the fact that in each time slot we have at most one project: for each time slot $t_r \in T$ and each pair of projects $p_j, p_{j'} \in P$ with $j \neq j'$ we add the clause $\neg x_{jr} \vee \neg x_{j'r}$. Let the conjunction of these clauses be C'.

Finally, for each agent $i \in [n]$, we create a clause that requires this agent to have positive utility. Specifically, for each $i \in [n]$ we create a clause C_i as follows. If there exists a single time slot t_r such that $s_{i,r} \neq \varnothing$, we set $C_i = x_{jr}$ if $s_{i,r} = \{p_j\}$ and $C_i = x_{jr} \vee x_{j'r}$ if $s_{i,r} = \{p_j, p_{j'}\}$. If there exists two time slots t_r, $t_{r'}$ such that $s_{i,r}, s_{i,r'} \neq \varnothing$, then it has to be the case that $s_{i,r} = p_j$, $s_{i,r'} = p_{j'}$ for some $p_j, p_{j'} \in P$, so we set $C_i = x_{jr} \vee x_{j'r'}$.

It is now easy to see that there exists a truth assignment that satisfies C^*, C', and all clauses in $C_1, \ldots, C_n$ if and only if there exists an outcome that guarantees positive utility to each agent. Moreover, each of the clauses in our construction is a disjunction of at most two literals. It remains to observe that 2-SAT is solvable in $\mathcal{O}(n + m)$ time [3, 11, 19]. □

4.3 Equitability

In Section 2, we have seen that not all instances admit equitable outcomes. We will now show that deciding the existence of equitable outcomes is computationally intractable. The proof is deferred to the appendix.

EQUITABILITY (EQ):
Input: A problem instance $I = (P, T, (\mathbf{s}_i)_{i \in N})$, and a parameter $\lambda \in \mathbb{N}$.
Question: Is there an outcome $\mathbf{o}$ such that $u_i(\mathbf{o}) = \lambda$ for each $i \in N$?

Theorem 4.6. EQ *is* NP*-complete. The hardness result holds even in the* γ*-PP setting, for any* γ*.*

4.4 Proportionality

Finally, we consider the complexity of finding proportional outcomes.

PROPORTIONALITY (PROP):
Input: A problem instance $I = (P, T, (\mathbf{s}_i)_{i \in N})$.
Question: Is there an outcome $\mathbf{o}$ such that $u_i(\mathbf{o}) \geq \mu_i/n$ for each $i \in N$?

It is easy to see that PROP does not necessarily imply UTIL or EGAL. Indeed, consider the case where $n = m = \ell$, and all agents have the same preference: $s_{i,j} = p_j$ for all $i \in [n]$, $j \in [\ell]$. Then, the only outcome that maximizes utilitarian or egalitarian social welfare is $(p_1, \ldots, p_m)$. However, any outcome with just a single project scheduled at the "correct" time slot would be a PROP outcome.

For proportionality, we obtain the following result. The proof can be found in the appendix.

Theorem 4.7. PROP *is at least as hard as* PCBP-RM*. The hardness result holds even in the FP setting.*

5 Conclusion and Future Work

We considered a variety of welfare and fairness objectives in the assignment of projects to time slots based on preferences of multiple agents. In particular, we showed that when the number of agents is bounded by a constant, most of the associated decision problems are solvable in polynomial time by a randomized algorithm. When the number of agents can be arbitrary, we obtain a polynomial-time algorithm for the utilitarian welfare, and hardness results for the egalitarian welfare, equitability and proportionality; for the egalitarian welfare, we also identify special cases where optimal outcomes can be computed efficiently.

Avenues for future research include the following: (1) relaxing the capacity constraints on time slots, so that we can implement multiple (or zero) projects at each time slot; (2) considering agents with different entitlements and the associated fairness notions [8, 12, 25, 34]; (3) exploring settings where agents belong to different groups and investigating group fairness and proportionality in our setting [9, 28]; (4) designing strategyproof scheduling mechanisms.

References

[1] Kolos Csaba Àgoston, Pèter Birò, and Richàrd Szàntò. Stable project allocation under distributional constraints. *Operations Research Perspectives*, 5:59–68, 2018.

[2] Ron Aharoni, Eli Berger, Dani Kotlar, and Ran Ziv. On a conjecture of Stein. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 87:203–211, 2017.

[3] Bengt Aspvall, Michael Plass, and Robert Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8: 121–123, 1979.

[4] Evripidis Bampis, Bruno Escoffier, and Sasa Mladenovic. Fair resource allocation over time. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 766–773, 2018.

[5] Steven Brams and Alan Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, 1996.

[6] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel Procaccia. *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

[7] James R Bunch and John E Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[8] Mithun Chakraborty, Ayumi Igarashi, Warut Suksompong, and Yair Zick. Weighted envy-freeness in indivisible item allocation. *ACM Transactions on Economics and Computation*, 9(3):18:1–18:39, 2021.

[9] Vincent Conitzer, Rupert Freeman, Nisarg Shah, and Jennifer Wortman Vaughan. Group fairness for the allocation of indivisible goods. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1853–1860, 2019.

[10] Edith Elkind, Neel Patel, Alan Tsang, and Yair Zick. Keeping your friends close: Land allocation with friends. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 318–324, 2020.

[11] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multicommodity flow problems. In *Proceedings of the 16th Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1975.

[12] Alireza Farhadi, Mohammad Ghodsi, MohammadTaghi HajiAghayi, Sébastien Lahaie, David M. Pennock, Masoud Seddighin, Saeed Seddighin, and Hadi Yami. Fair allocation of indivisible goods to asymmetric agents. *Journal of Artificial Intelligence Research*, 64:1–20, 2019.

[13] Moti Frances and Ami Litman. On covering problems of codes. *Theory of Computing Systems*, 30:113–119, 2007.

[14] Rupert Freeman, Seyed Majid Zahedi, and Vincent Conitzer. Fair and efficient social choice in dynamic settings. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4580–4587, 2017.

[15] Rupert Freeman, Seyed Majid Zahedi, Vincent Conitzer, and Benjamin C. Lee. Dynamic proportional sharing: A game-theoretic approach. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–36, 2018.

[16] Mariano Gasca and Thomas Sauer. Polynomial interpolation in several variables. *Advances in Computational Mathematics*, 12(4):377–410, 2000.

[17] David Karger, Cliff Stein, and Joel Wein. *Scheduling Algorithms*, page 20. Chapman & Hall/CRC, 2 edition, 2010. ISBN 9781584888208.

[18] Judy Kay and Piers Lauder. A fair share scheduler. *Communications of the ACM*, 31 (1):44–55, jan 1988.

[19] Melven Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1–2):15–20, 1967.

[20] Martin Lackner. Perpetual voting: Fairness in long-term decision making. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, pages 2103–2110, 2020.

[21] Martin Lackner, Jan Maly, and Simon Rey. Fairness in long-term participatory budgeting. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 299–305, 2021.

[22] Kevin Lanctot, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003.

[23] Andrea Lodi, Philippe Olivier, Gilles Pesant, and Sriram Sankaranarayanan. Fairness over time in dynamic resource allocation with an application in healthcare. *arXiv preprint arXiv:2101.03716*, 2022.

[24] Michael R Merrifield and Donald G Saari. Telescope Time Without Tears: A Distributed Approach to Peer Review. *Astronomy & Geophysics*, 50(4):4.16–4.20, 2009.

[25] Luisa Montanari, Ulrike Schmidt-Kraepelin, Warut Suksompong, and Nicholas Teh. Weighted envy-freeness for submodular valuations. *arXiv preprint*, arXiv:2209.06437, 2022.

[26] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[27] Guillem Perarnau and Oriol Serra. Rainbow matchings: Existence and counting. *arXiv preprint arXiv:1104.2702*, 2011.

[28] Jonathan Scarlett, Nicholas Teh, and Yair Zick. For one and all: Individual and group fairness in the allocation of indivisible goods. In *Proceedings of the 8th International Workshop on Computational Social Choice (COMSOC)*, 2021.

[29] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

[30] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[31] Alex Schwarzenberg-Czerny. On matrix factorization and efficient least squares solution. *Astronomy and Astrophysics Supplement Series*, 110:405, 1995.

[32] Colin T.S. Sng and David F. Manlove. Popular matchings in the weighted capacitated house allocation problem. *Journal of Discrete Algorithms*, 8(2):102–116, 2010.

[33] Philipp Ströhle, Enrico H. Gerding, Mathijs M. de Weerdt, Sebastian Stein, and Valentin Robu. Online mechanism design for scheduling non-preemptive jobs under uncertain supply and demand. In *Proceedings of the 24th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 437–444, 2014.

[34] Warut Suksompong and Nicholas Teh. On maximum weighted Nash welfare for binary valuations. *Mathematical Social Sciences*, 117:101–108, 2022.

[35] Richard Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9(3):375–403, 1990.

Edith Elkind
University of Oxford
Oxford, United Kingdom
Email: `elkind@cs.ox.ac.uk`

Sonja Kraiczy
University of Oxford
Oxford, United Kingdom
Email: `sonja.kraiczy@cs.ox.ac.uk`

Nicholas Teh
University of Oxford
Oxford, United Kingdom
Email: `nicholas.teh@cs.ox.ac.uk`

6 Appendix

6.1 Proof of Proposition 3.5: PROP1 outcome exists for two agents.

Proof. For $i = 1, 2$, let $M_i \subset E_i$ be some maximum matching in G_i. We place each edge $e \in M_1 \cap M_2$ in M; note that each such edge does not share endpoints with other edges in $M_1 \cup M_2$. Let $\kappa = |M_1 \cap M_2|$, and for $i = 1, 2$, set $\mu_i' = |M_i \setminus M_{3-i}| = \mu_i - \kappa$. Consider the graph $G' = (V, (M_1 \cup M_2) \setminus (M_1 \cap M_2))$. We will say that an edge e of G' is *red* if $e \in M_1$ and *blue* if $e \in M_2$; note that each edge of G' is either red or blue. Moreover, in G' each vertex is incident on at most one red edge and at most one blue edge, so each connected component of G' is an isolated vertex, an alternating red-blue path, or an alternating red-blue cycle (which must be of even length).

Next, we construct an instance of the fair allocation problem with two agents $N = \{1, 2\}$ and divisible items. We create one item for each connected component of G'; let I be the resulting set of items. For each $C \in I$ we define the valuation of agent $i \in \{1, 2\}$ to be $v_i(C) = |E(C) \cap M_i|$. We extend the valuation function to bundles and fractional allocations in a linear fashion: if agent i obtains a bundle that contains an α_C-fraction of item C for $C \in I'$, then her valuation for this bundle is $\sum_{C \in I'} \alpha_C \cdot v_i(C)$. Note that we have $v_i(I) = \mu_i'$.

We then apply the adjusted winner procedure [5] to this instance. This procedure results in a proportional allocation that splits at most one of the items in I. For $i = 1, 2$, let I_i be the set of items that are allocated to agent i, and let C^* be the item that is split, so that agent i gets a w_i-fraction of C^*, where $w_1 + w_2 = 1$. By proportionality, the value of agent i for her bundle, given by $\sum_{C \in I_i} v_i(C) + w_i \cdot v_i(C^*)$, is at least $\mu_i'/2$. We will now transform this allocation into a matching in G'.

To this end, for each $i = 1, 2$, if the connected component C is allocated to i, we place edges $M_i \cap C$ into M. To select edges from C^*, we proceed as follows.

Suppose first that C^* is a path of length $2s$, with red edges $e_1, \ldots, e_{2s-1}$ and blue edges $e_2, \ldots, e_{2s}$. Then $v_1(C^*) = v_2(C^*) = s$. Let $t = \lfloor w_1 \cdot s \rfloor$, and place edges $e_1, \ldots, e_{2t-1}$ as well as $e_{2t+2}, \ldots, e_{2s}$ into M: by construction, these edges form a matching, which contains $t \geq w_1 \cdot s - 1$ red edges and $s - t \geq s - w_1 \cdot s = w_2 \cdot s$ blue edges.

Next, suppose that C^* is a path of length $2s - 1$, with red edges $e_1, \ldots, e_{2s-1}$ and blue edges $e_2, \ldots, e_{2s-2}$. Then $v_1(C^*) = s$, $v_2(C^*) = s - 1$. Again, let $t = \lfloor w_1 \cdot s \rfloor$, and place edges $e_1, \ldots, e_{2t-1}$ as well as $e_{2t+2}, \ldots, e_{2s-2}$ into M: by construction, these edges form a matching, which contains $t \geq w_1 \cdot s - 1$ red edges and $s - t - 1 \geq s - w_1 \cdot s - 1 = w_2 \cdot s - 1$ blue edges.

Finally, if C^* is a cycle of length $2s$, we discard one blue edge, transforming C^* into an odd-length path, and then proceed as described in the previous paragraph, i.e., select at least $w_1 \cdot s - 1$ red edges and $w_2 \cdot s - 1$ blue edges and place them in M. This completes the construction of M; it is immediate that it can be carried out in polynomial time.

For each item $C \in I_1$, we place $|E(C) \cap M_1| = v_1(C)$ red edges in M, and, when splitting C^*, we place at least $w_1 \cdot v_1(C^*) - 1$ red edges in M. By proportionality of the fractional allocation, we have $\sum_{C \in I_1} v_1(C) + w_1 \cdot v_1(C^*) \geq \mu_1'/2$; hence, we place at least $\mu_1'/2 - 1$ red edges in M. By a similar argument, we place at least $\mu_2'/2 - 1$ blue edges in M. In addition, M contains κ edges from $M_1 \cap M_2$. Hence, it contains at least $\mu_i'/2 - 1 + \kappa = (\mu_i - \kappa)/2 - 1 + \kappa \geq \mu_i/2 - 1$ edges from M_i for each $i = 1, 2$. □

6.2 Proof of Theorem 4.6: Eq is *NP*-complete. The hardness result holds even in the γ-PP setting, for any γ.

Proof. It is clear that Eq is in NP. To show that this problem is NP-hard, we first formulate an intermediate problem to be used in our proof. Namely, we introduce the Exact Partial Binary Closest String Problem (Exact-P-BCSP). This problem is similar to the BCSP, but with two differences: (1) the Hamming distance between the output string and each of the input strings must be exactly κ, and (2) we allow an additional character, $*$, in the solution string. Formally, an instance of Exact-P-BCSP consists of ν binary strings of length ρ each, and an integer κ; it is a yes-instance if there exists a string y of length ρ over the alphabet $\{0, 1, *\}$ such that the Hamming distance between y and each of the ν input strings is exactly κ (equivalently, the number of matches is exactly $\rho - \kappa$), and a no-instance otherwise.

We begin with the following lemma, whose proof we include separately.

Lemma 6.1. Exact-P-BCSP *is* NP*-hard.*

We will now reduce Exact-P-BCSP to Eq. Consider an instance of Exact-P-BCSP given by ν binary strings $X = \{x_1, \ldots, x_\nu\}$ of length ρ each and an integer κ. For each $i \in [\nu]$, $j \in [\rho]$, denote the j-th bit of the i-th string by x_{ij}. We introduce 2ρ projects $p_1, \ldots, p_{2\rho}$ and 2ρ time slots $t_1, \ldots, t_{2\rho}$. We encode the bit strings in X as the agents' preferences: for each $i \in [\nu]$, $j \in [\rho]$, if $x_{ij} = 1$, let $s_{i,2j-1} = p_j$, $s_{i,2j} = \varnothing$; and if $x_{ij} = 0$, let $s_{i,2j-1} = \varnothing$, $s_{i,2j} = p_j$.

We will now prove that there exists an outcome $\mathbf{o}$ that gives each agent a utility of exactly $\rho - \kappa$ if and only if there exists a binary string y of length ρ such that the number of mismatches is exactly κ.

For the 'if' direction, let y be a solution string with exactly $\rho - \kappa$ matches to each of the strings in X. We construct an outcome $\mathbf{o}$ as follows. For each $j \in [\rho]$, if $y_j = 1$ we let $o_{2j-1} = p_j$ and if $y_j = 0$ we let $o_{2j} = p_j$; we assign the remaining projects to the remaining slots arbitrarily, with the constraint that if $y_j = *$, then p_j is not assigned to either of the

time slots t_{2j-1}, t_{2j} (it is not difficult to verify that this can always be done efficiently; we omit the details). Consider an agent $i \in [\nu]$ and a time slot $j \in [\rho]$. If $x_{ij} = y_j = 1$ we have $o_{2j-1} = s_{i,2j-1}$, $s_{i,2j} = \varnothing$, if $x_{ij} = y_j = 0$ we have $o_{2j} = s_{i,2j}$, $s_{i,2j-1} = \varnothing$, and if $x_{ij} \neq y_j$ then $o_{2j-1} \notin s_{i,2j-1}$, $o_{2j} \notin s_{i,2j}$. Hence, each pair of time slots (t_{2j-1}, t_{2j}) such that $x_{ij} = y_j$ contributes exactly 1 to the utility of agent i, so $u_i(\mathbf{o}) = \rho - \kappa$.

For the 'only if' direction, consider any outcome $\mathbf{o}$ that gives each agent a utility of exactly $\rho - \kappa$. To construct the string y, for each $j \in [\rho]$ we set

$$y_j = \begin{cases} 1 & \text{if } o_{2j-1} = p_j \\ 0 & \text{if } o_{2j} = p_j \\ * & \text{otherwise} \end{cases}$$

Consider an agent i. Observe that her utility from the pair of time slots (t_{2j-1}, t_{2j}) is at most 1; moreover, it is 1 if and only if (1) $s_{i,2j-1} = o_{2j-1} = p_j$ or (2) $s_{i,2j} = o_{2j} = p_j$. Condition (1) holds if and only if $x_{ij} = y_j = 1$, and condition (2) holds if and only if $x_{ij} = y_j = 0$. That is, agent i's utility from (t_{2j-1}, t_{2j}) is 1 if and only if $x_{ij} = y_j$. Since we have $u_i(\mathbf{o}) = \rho - \kappa$ for each $i \in [\nu]$, this means that y has κ mismatches with every input string. □

6.3 Proof of Lemma 6.1: Exact-P-Bcsp is NP-hard.

Proof. We prove the claim by reduction from Exact-2-3SAT (X2-3SAT). In an instance of X2-3SAT, we are given a 3-CNF formula with n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. In each clause, exactly two literals are evaluated to True. Note that X2-3SAT is equivalent to Exact-1-3SAT (where exactly one literal is evaluated to True), and the latter is known to be NP-hard [29].

Consider an instance of X2-3SAT, given by n variables and m clauses, each with at most 3 literals. We will encode the Boolean value assignment to variables in the bits of the strings. Consider strings of length $2n + 1$; for all $i \in [n]$, if $x_i = 1$, let $s_{i,2j-1} = 1$ and $s_{i,2j} = 0$; if $x_i = 0$, let $s_{i,2j-1} = 0$ and $s_{i,2j} = 1$.

Then, first create a string of the form $(00)^n 1$ and n strings of the form $(00)^i 11 (00)^{n-i-1} 1$, for all $i \in [n-1]$. We mandate that the distance from the solution string be of hamming distance n from each of these strings. We derive the following claims.

Claim 1: Each consecutive pair of bits $s_{i,2j-1}$ and $s_{i,2j}$ only admit bit combinations of the form 10 or 01. Equation that arises from the first string is

$$(1 - x_{2n+1}) + \sum_{j \in [2n]} x_j = n \tag{1}$$

Equation that arises from the i-th string (for $i = 2, \ldots, n+1$) is,

$$(1 - x_{2n+1}) + (1 - x_{2i-3}) + (1 - x_{2i-2}) + \sum_{j \in [2i] \setminus \{2i-3, 2i-2\}} x_j = n \tag{2}$$

Now, for each $i = 2, \ldots, n+1$, subtract (1) from (2), we get

$$1 - x_{2i-3} - x_{2i-3} + 1 - x_{2i-2} - x_{2i-2} = 0. \tag{3}$$

Dividing both sides of the equation by 2, we get

$$x_{2i-3} + x_{2i-2} = 1, \tag{4}$$

indicating that the bits of each pair should be 01 or 10.

Claim 2: The last bit must be 1. Suppose, for a contradiction, that the last bit of the solution is 0. This means that the first string has distance $n-1$ for the first $2n$ bits, and by the pigeonhole principle, at least one pair will be 00. Let such a pair be (s_{2i-1}, s_{2i}) for some $i \in [n]$. However, if we consider the $(i+1)$-th string, where the $(2i-1)$-th and $(2i)$-th bit is 11, the distance of any solution to this (by Claim 1) has to be $n-1$ (for the $n-1$ remaining pairs of 00) + 2 (for the mismatch mentioned above) + 1 (for the last bit) $= n+2 > n$, contradicting the fact that the distance has to be exactly n.

Next, for each of the m clauses, we create a string whose bits depends on what truth assignment to the variable(s) in the clause would make the literal positive: i.e. if the literal appears and is positive, we let the corresponding pair be 10, if it is negative, we let it be 01, and if it doesn't appear in the clause, we let it be 00. Also, let the last bit be 0. We mandate that the distance of the solution string to each of these strings be n.

Note that an implication of Claim 2 is that the null character cannot appear in the solution string.

Claim 3: There must be exactly two pairs of matching consecutive bit pairs. From Claim 2 above, the last bit must be 1, so we have a hamming distance of 1 from there. For $n-3$ pairs corresponding to literals not in the clause, the distance to each must be 1 (by Claim 1 above), thereby giving us a hamming distance of $n-3$. Then, we have three pairs remaining with total hamming distance of 2 (since the total distance must be n). Again by Claim 1 above, it must be that exactly two pairs match and one pair does not.

We are left with proving that there exists a binary string y of length m such that the number of mismatches is exactly $m-n$ if and only if there exists an assignment of Boolean values to variables such that exactly two literals in each clause evaluates to True.

By the three claims above, it is easy to see that the functions mapping the solutions between the problem are the same (by looking at the first $2n$ time slots in Exact-P-Bcsp). □

6.4 Proof of Theorem 4.7: Prop is at least as hard as PCBP-RM in the FP setting.

Proof. We prove the claim by reduction from Egal with $\lambda = 1$. An instance of Egal with $\lambda = 1$ consists of n agents, m projects, m time slots, and an integer λ; it is a yes-instance if there exists an outcome $\mathbf{o}'$ such that every agent gets a utility of at least 1; and a no-instance otherwise.

Consider such an instance of Egal with $\lambda = 1$ given by n agents $N = \{1, \ldots, n\}$, m projects $P = \{p_1, \ldots, p_m\}$ and m time slots $T = \{t_1, \ldots, t_m\}$. Now, make copies of all agents and their approval sets. We make only one modification: if $n < m$, then duplicate the number of agents until its equal to m. Let the number of agents in this new instance be n' Thus, we have that $n' \geq m$. Note that an outcome is a solution to the Egal problem with the original n agents if and only if the same outcome is a solution to the Egal problem with n' agents (i.e., the only change to the problem duplicating agents does is changing the number of agents).

Next, we will prove that there exists an outcome $\mathbf{o}$ that gives each agent at least a utility of $\frac{m}{n'}$ if and only if there exists an outcome $\mathbf{o}'$ that gives each agent at least a utility of 1.

For the 'if' direction, let $\mathbf{o}'$ be an outcome that guarantees each agent at least a utility of 1. Then, since $n' \geq m$, $\frac{m}{n'} \leq 1$ and hence, any outcome that guarantees each agent a utility of 1 satisfies proportionality, i.e., let $\mathbf{o} = \mathbf{o}'$.

For the 'only if' direction, let $\mathbf{o}$ be an outcome that guarantees each agent at least a utility of $\frac{m}{n'}$. Then, since the utility of an agent is an integer, the outcome $\mathbf{o}$ guarantees each agent at least a utility of 1, i.e. let $\mathbf{o}' = \mathbf{o}$.

□