Fair Coordination in Strategic Scheduling

Wei-Chen Lee
University of Oxford
Oxford, United Kingdom
wei-chen.lee@cs.ox.ac.uk

Martin Bullinger
University of Bristol
Bristol, United Kingdom
martin.bullinger@bristol.ac.uk

Alessandro Abate
University of Oxford
Oxford, United Kingdom
alessandro.abate@cs.ox.ac.uk

Michael Wooldridge
University of Oxford
Oxford, United Kingdom
michael.wooldridge@cs.ox.ac.uk

ABSTRACT

We consider a scheduling problem of strategic agents representing jobs of different weights. Each agent has to decide on one of a finite set of identical machines to get their job processed. In contrast to the common, isolated focus on makespan minimization, we incorporate fairness with respect to the strategic considerations of the agents. Two natural properties are credibility, which ensures that the assignment is a Nash equilibrium and equality, requiring that agents with equal-weight jobs are assigned to machines of equal load. We combine these two with a hierarchy of fairness properties based on envy-freeness together with several relaxations based on the idea that envy seems more justified towards agents with a higher weight. We present a complete complexity landscape for satisfiability and decision versions of these properties, alone or in combination, and study them as structural constraints under makespan optimization. For our positive results, we develop a unified algorithmic approach, where we achieve different properties by fine-tuning key subroutines.

KEYWORDS

Non-cooperative game theory; Coordination; Fairness; Scheduling; Load balancing

ACM Reference Format:
Wei-Chen Lee, Martin Bullinger, Alessandro Abate, and Michael Wooldridge. 2026. Fair Coordination in Strategic Scheduling. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/BOCM2167

1 INTRODUCTION

We study a strategic variant of the classic *identical machine scheduling problem* [10], where each of n weighted tasks is assigned to one of m identical resources (machines). The load on a resource is the sum of the weights of the tasks assigned to it, and the objective is to minimize the largest load on any resource, known as the 'makespan' of the assignment. The interpretation is to consider the weight of a task as the time taken to process it, and thus the makespan is the time required to process all tasks. Strategic variants of this problem, where each task is owned by a different player concerned only with the completion time of their own task, or the completion time of the resource to which their task is assigned, are known as the *selfish load balancing problem* [20] or the *weighted singleton congestion game* [16]. In this paper, we adopt the latter objective, as the resource load more closely captures the notion of congestion.

In a strategic variant where players can coordinate their choices, fairness is an important consideration. A natural idea in this context is envy-freeness [7, 19], a standard notion in fair division [4], which requires that no agent would rather replace another agent on their assigned resource. However, naive envy-freeness is often impossible to satisfy (see Figure 2 for an example) and even feels unnecessarily rigid as agents congest machines differently. This motivates the relaxation of envy-freeness where envy is only justified towards agents representing a heavier job. We introduce a hierarchy of such notions and study them along two other natural desiderata: credibility, a Nash equilibrium condition that demands that no player can benefit from a unilateral deviation, and equality, another fairness notion that requires agents of equal weight to be assigned to machines of equal load. We then consider the algorithmic challenge of computing desirable assignments that satisfy single or several of these properties.

Notably, our properties can be seen as natural constraints for the identical machine scheduling problem. Interestingly, (the combination of) several of them render this problem polynomial-time solvable. We view it as an important message from our paper that natural constraints can facilitate computational feasibility.

1.1 Related Literature

Early works in the scheduling literature establish that the base problem of identical machine scheduling (without further constraints) is NP-complete [9], and that a polynomial-time approximation of $\frac{4}{3} - \frac{1}{3m}$ can be achieved using the *Longest Processing Time First (LPT)* algorithm [10].

Strategic variants of this problem were introduced subsequently. [16, 20]. These build on important properties of congestion games, such as the finite improvement property, existence of pure-strategy Nash equilibria, or the correspondence with potential games established in [17, 18]. The LPT algorithm was found to provide a pure-strategy Nash equilibrium [8]. Subsequent work on the price of anarchy of atomic congestion games (e.g., [1–3, 5, 6, 12]) studied the efficiency gap between a worst equilibrium and an efficient outcome.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/BOCM2167

Table 1: Summary of the satisfiability and complexity for combinations of various fairness and credibility conditions. For definitions see Section 2. In the third and fourth column, we either give a polynomial bound on the running time for computing a satisfying assignment with or without a makespan constraint, or state NP-completeness of deciding on existence.

Property P	Exists	Unique	P-Satisfiability	P-Makespan
$\top$	Y	N	$O(n)$	NP-complete [9]
Eq	Y	N	$O(n)$	NP-complete
Cr	Y	N	$O(n \log n)$ [8]	NP-complete
$Eq \wedge Cr$	N	N	NP-complete	NP-complete
EF	N	N	$O(n\sqrt{n})$	$O(n\sqrt{n})$
$EF \wedge Cr$	N	Y	$O(n\sqrt{n})$	$O(n\sqrt{n})$
WOE	Y	N	$O(n)$	$O(n \log n)$
$WOE \wedge Eq \equiv OE$	Y	N	$O(n)$	$O(n^2)$
$WOE \wedge Cr$	N	Y	$O(n \log n)$	$O(n \log n)$
$WOE \wedge Eq \wedge Cr \equiv OE \wedge Cr$	N	Y	$O(n \log n)$	$O(n \log n)$
SM	N	N	$O(n \log n)$	$O(n \log n)$
$SM \wedge Eq$	N	N	$O(n \log n)$	$O(n \log n)$
$SM \wedge Cr$	N	Y	$O(n \log n)$	$O(n \log n)$
$SM \wedge Eq \wedge Cr$	N	Y	$O(n \log n)$	$O(n \log n)$
WM	Y	N	$O(n)$	NP-complete
$WM \wedge Eq \equiv M$	Y	N	$O(n)$	NP-complete
$WM \wedge Cr$	N	N	NP-complete	NP-complete
$WM \wedge Eq \wedge Cr \equiv M \wedge Cr$	N	N	NP-complete	NP-complete

Our paper is most closely related to recent work on communication partition [13, 14], where players are partitioned into coalitions and communicate locally within their coalition. They play a correlated action profile (assignment) if a credible, efficient, and fair agreement can be reached within each coalition. We expand on this work in two important dimensions. First we relax the arguably restrictive envy-freeness property by considering a hierarchy of weaker notions of fairness; second, rather than characterizing the structure of desirable outcomes, we study the computational complexity of finding an assignment for a coalition of players under all combinations of such axiomatic properties.

1.2 Our Contribution

A summary of our results is provided in Table 1. We answer the following four questions for each (single or composite) property P:

- Existence: Does a P-satisfying assignment always exist?
- Uniqueness: If a P-satisfying assignment exists, is it unique?
- P-Satisfiability: What is the computational complexity of finding a P-satisfying assignment?
- P-Makespan: What is the computational complexity of finding a P-satisfying assignment of smallest makespan?

We find that, for certain classes of this decision problem, it is sufficient to search for a 'contiguous' assignment which limits the search space and avoids the combinatorial blow-up. However, especially when this is not possible, we provide a reduction to show that they are indeed NP-complete.

For the remainder of this paper, we formally define the problem, our notation, and the various properties of interest in Section 2. We present tractability results (computable in polynomial time) in Section 3, and intractability results (NP-completeness) in Section 4. We conclude in Section 5. An extended version of our paper containing formal proofs, all subroutines of our main algorithm, and results concerning envy-freeness is available on arXiv [15].

2 PRELIMINARIES

For positive integers $k, \ell \in \mathbb{Z}_{>0}$, we denote $[k] := \{1, \dots, k\}$ and $[k : \ell] := \{k, \dots, \ell\}$. Given a vector $\mathbf{v}$, we denote by $|\mathbf{v}|$ its length. We have a set $[n]$ of n weighted *players*, and a set $[m]$ of m identical *resources*. The weights of players are represented by a vector $\mathbf{w} = (w_i)_{i \in [n]} \in \mathbb{Z}^n_{>0}$ of positive integers. We denote by $\mathrm{supp}(\mathbf{w})$ the set of unique weights in $\mathbf{w}$, and for $w \in \mathrm{supp}(\mathbf{w})$, we set $\#(w) := |\{i \in [n] : w_i = w\}|$ the number of players of weight w with respect to $\mathbf{w}$. An assignment is an n-tuple $\mathbf{a} = (a_1, \dots, a_n)$ where $a_i \in [m]$ specifies the resource that is assigned to player i.

We can also express $\mathbf{a}$ in terms of the *load distribution* on each resource. For example, consider the instance $\mathbf{w} = (4, 3, 3, 1)$, $m = 3$, and assignment $\mathbf{a} = (1, 1, 2, 2)$, leaving the third resource unused. The load distribution can be written as $l(\mathbf{a}) = \{\{4, 3\}, \{3, 1\}, \varnothing\}$, a multiset whose components are multisets. This representation groups equivalent assignments, only differing between players of same weight, into the same load expression. Two assignment are said to be equivalent if they have the same load distribution, i.e., $\mathbf{a} \equiv \mathbf{a}' \iff l(\mathbf{a}) = l(\mathbf{a}')$.

Given an assignment $\mathbf{a}$, the *load* on some resource $x \in [m]$ is the sum of all the players' weights assigned to it, denoted as $v_x(\mathbf{a}) := \sum_{i:a_i=x} w_i$, and is the cost incurred by each player assigned to x.[1] The makespan of the assignment is the maximum load on any machine, i.e. $c(\mathbf{a}) := \max_{x \in [m]} v_x(a)$. For brevity and consistency, we use the variables $i, j, k \in [n]$ for player indices or counts, and the variables $x, y \in [m]$ for resource indices or counts. Additionally, $W := \sum_i w_i$ denotes the total weight of all players, $\mathbf{w}[i : j] := (w_i, ..., w_j)$ denotes a contiguous list of sorted weights, and $W[i : j] := \sum_{k=i}^{j} w_k$ denotes the sum of weights of such a contiguous set of players.

2.1 Problem Definition

The problem we study can be formulated in a number of equivalent ways. We define it as a constrained identical machine scheduling problem to emphasize the connection of our computational complexity results to the known ones for the classic identical machine scheduling problem.

Given a (single or composite) property P our main problem is to minimize makespan subject to satisfying this property.

Problem: P-Makespan
Input: A weight vector $\mathbf{w} = (w_1, ..., w_n)$ of positive integers, a positive integer m, makespan threshold t.
Task: Find a P-satisfying assignment of makespan at most t, or decide that no such assignment exists.

Throughout the paper, we assume that $n > m > 1$ (i.e., there are more players than resources, and more than one resource), and that $\max_i w_i \leq t$. The case $n \leq m$ is rather trivial: an assignment where each player is assigned a different resource satisfies all of

[1] Equivalently, this is a weighted singleton congestion game with a linear cost function.

the conditions considered in this paper, and we can answer P-Makespan by checking whether the largest weight in $\mathbf{w}$ exceeds the makespan threshold t. Moreover, the case $m = 1$ allows only for a single assignment (all players assigned to the same resource), which can be checked to solve P-Makespan. Finally, the case $\max_i w_i > t$ clearly has no solution.

We also investigate the problem of satisfying P in isolation, as defined next. It can be formulated as a special case of P-Makespan with $t = \infty$ or $t = W$.

Problem: P-Satisfiability
Input: A weight vector $\mathbf{w} = (w_1, ..., w_n)$ of positive integers and a positive integer m.
Task: Find a P-satisfying assignment, or decide that no such assignment exists.

2.2 Properties and Their Relations

The properties that we consider are defined as follows. Recall that v_{a_i} is the load on the resource assigned to player i.

Given an assignment $\mathbf{a}$, we say that $\mathbf{a}$ satisfies the following property if for all $i, j \in [n], x \in [m]$:

- *Credibility (Cr)*: $v_{a_i} \leq v_x + w_i$.
- *Equality (Eq)*: $w_i = w_j \implies v_{a_i} = v_{a_j}$.
- *Envy-freeness (EF)*: $v_{a_i} - w_i \leq v_{a_j} - w_j$.
- *Weak ordered envy-freeness (WOE)*: $w_i < w_j, a_i \neq a_j \implies v_{a_i} - w_i \leq v_{a_j} - w_j$.
- *Ordered envy-freeness (OE)*: $w_i \leq w_j, a_i \neq a_j \implies v_{a_i} - w_i \leq v_{a_j} - w_j$.
- *Strong monotonicity (SM)*: $w_i < w_j \implies v_{a_i} < v_{a_j}$.
- *Weak monotonicity (WM)*: $w_i < w_j \implies v_{a_i} \leq v_{a_j}$.
- *Monotonicity (M)*: $w_i \leq w_j \implies v_{a_i} \leq v_{a_j}$.

Single properties can be combined to form composite properties, e.g., $\mathbf{a}$ satisfies $WOE \wedge Eq$ means $\mathbf{a}$ satisfies *WOE* and $\mathbf{a}$ satisfies *Eq*. We use the symbol $\top$ to denote the empty conjunction of properties, i.e., there are no constraints. We speak of a P-assignment to refer to an assignment satisfying P. *Cr* is the property that no player can benefit from a unilateral deviation, or equivalently the agreement is a pure-strategy Nash equilibrium, which ensures that the non-binding agreement is honored by the players. *EF* is a strong fairness property that requires no player to want to switch their assignment with another. *WOE* is a weaker version of *EF*, which only requires the lighter player of any pair of players to not envy the assignment of the heavier player. The condition applies only to pairs of players that are assigned different resources, since it would make no sense for a player to envy another player assigned to the same resource. *SM* is another fairness condition that says a heavier player should incur more cost than a lighter player, while *WM* says that a heavier player should incur no less cost than a lighter player. Since *WOE*, *SM*, and *WM* do not impose any fairness conditions on players of the same weight, we also introduce *Eq*, which requires players of the same weight to incur the same cost. Finally, *OE* and *M* combine the *Eq* condition with *WOE* and *WM* respectively, as we show in Proposition 3.

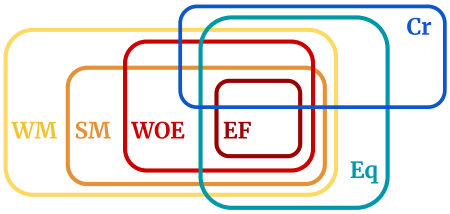


Figure 1: Venn diagram of relations between properties.

Remark 1 (Other properties). *[13, 14] additionally consider Pareto optimality; however, since it is implied by the stronger condition of credibility in our model, and is trivially satisfied by any assignment that uses all resources (in the case $n > m$), we do not separately consider it in this paper. Also notice that we do not define 'Strong ordered envy-freeness' (SOE), which would logically be the condition $v_{a_i} - w_i < v_{a_j} - w_j$ for all i, j with $w_i < w_j, a_i \neq a_j$. This is due to the fact that this is an unreasonably strong condition which cannot be satisfied even in trivial cases. For example, no SOE-assignment exists for the instance $\mathbf{w} = (2, 3), m = 2$; even the assignment where $l(\mathbf{a}) = \{\{3\}, \{2\}\}$ leads player of weight 2 to strongly envy the player of weight 3.*

Remark 2 (Equivalent problems). *The $\top$-Makespan problem, where no constraint is applied, is equivalent to the classic identical machine scheduling problem [10]. The Cr-Satisfiability problem is equivalent to finding a pure-strategy Nash equilibrium in the singleton congestion game [6], and the Cr-Makespan problem can be used to find the socially optimal pure-strategy Nash equilibrium via a binary search for the optimal t.*

The relations between our properties are set out in Proposition 3. Figure 1 illustrates them with a Venn diagram.

Proposition 3. *[Property relations] The following relations hold:*

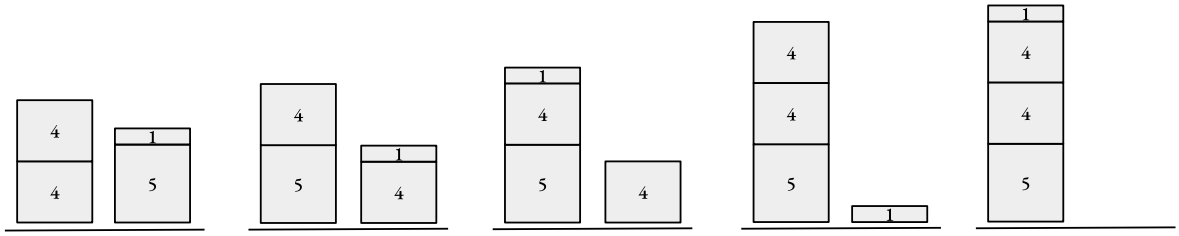
(1) $EF \implies Eq$
(2) $EF \implies WOE \implies WM$
(3) $EF \implies SM \implies WM$
(4) $WOE \wedge Eq \iff OE$
(5) $WM \wedge Eq \iff M$

While it might seem natural that *WOE* would imply *SM*, this is not the case, because *WOE* does not consider envy between players assigned to the same resource. Examples of assignments satisfying various properties are shown in Figure 2.

2.3 Contiguous Assignment

An important property that the satisfying assignments of many (conjunctions of) properties have is contiguity as defined next. This property will allow us to quickly search for a satisfying assignment in its canonical ordering.

An assignment is said to be *contiguous* if under a non-increasing ordering of player weights where players are indexed such that $i < j \implies w_i \geq w_j$, we have $(i < j) \wedge (a_i = a_j) \implies a_i = a_k \; \forall k \in [i : j]$. Assume we are given a contiguous assignment where players are indexed in a non-increasing order satisfying the contiguity definition. Then, we can further index the resources in a non-increasing order such that, for all $i, j \in [n], w_i \geq w_j \implies a_i \leq$



l(a), c(a)	{{4,4},{5,1}}, 8	{{5,4},{4,1}}, 9	{{5,4,1},{4}}, 10	{{5,4,4},{1}}, 13	{{5,4,4,1},∅}, 14
Contiguous?	No	Yes	No	Yes	Yes
Satisfied	Cr, Eq	Cr, WOE, WM	∅	WOE, WM, Eq	EF, WOE, WM, Eq
Not satisfied	EF, WOE, SM, WM	EF, SM, Eq	Cr, EF, WOE, SM, WM, Eq	Cr, EF, SM	Cr, SM

Figure 2: Examples of assignments for an instance where $\mathbf{w} = (5, 4, 4, 1)$, $m = 2$.

a_j. We call such an ordering of players and resources a *canonical ordering* of a contiguous assignment.

Thus, under the canonical ordering of a contiguous assignment, the first contiguous set of players are assigned to the first resource, the second contiguous set of players to the second resource, etc. The search for a contiguous assignment only has to find the right 'cuts' to make in order to assign players to resources, which significantly reduces the computational complexity of the problem.

Example 4. *Consider the load multiset* $l(\mathbf{a}) = \{\{2, 3\}, \{3, 3\}, \{1, 2\}\}$. *This is a contiguous assignment since we can order the player weights as* $(3, 3, 3, 2, 2, 1)$. *We can then index the resources to express the assignment as* $\{\{3, 3\}, \{3, 2\}, \{2, 1\}\}$ *in its canonical ordering.*

3 TRACTABLE PROPERTIES

In this section, we aim at algorithmic feasibility. We present a general algorithmic framework based on Algorithm 1, a polynomial-time algorithm that we term *Sequential Contiguous Assignment (SCA)*. It exploits the fact that a P-assignment is necessarily contiguous for a wide range of properties. The algorithm does so by searching for a contiguous assignment in its canonical ordering. We start by setting out the general, modular structure of the algorithm, and then specify the subroutines corresponding to various properties P in subsequent sections.

Players are first sorted in a non-increasing order. Then the algorithm iteratively finds the next number of players k to assign to a new resource. It proceeds until either a satisfying assignment is found, or it runs out of resources before all players are assigned, in which case no satisfying assignment is found. A final check is conducted for some cases of P to ensure that the candidate assignment does indeed satisfy P. The subroutines First_k, Subsequent_k and Satisfies differ for different problems, and are specified in the dedicated sections. Note that a subroutine might also return None, which causes termination of the algorithm.

We make a few general observations about the algorithm: First, the makespan threshold t is updated after each iteration to be the load of the resource that was just assigned to ensure a non-increasing ordering of load, which is necessary for certain fairness conditions.

Second, after the assignment of the first k players to the first resource, Subsequent_k is repeatedly called with the remaining players and resources until all players or resources are assigned. An additional argument $\mathcal{I}$ is computed and passed to the Subsequent_k subroutine; $\mathcal{I}$ contains information about the players assigned to the previous resource (e.g., maximum weight, minimum weight, number of players), which is used by Subsequent_k to ensure the satisfaction of P in the current assignment.

Finally, the algorithm allows for only a single value of k to be returned by the First_k and Subsequent_k subroutines at each iteration. This is sufficient as the algorithm searches for the assignment that is 'closest' to satisfying P, so that if no assignment is found, then no satisfying assignment exists.

Algorithm 1 Sequential Contiguous Assignment (SCA)

Input: A weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, a number of resources m, a property P, and a makespan threshold t.
Output: A contiguous P-assignment of makespan at most t, if it exists; None otherwise.

Sort and index $\mathbf{w}$ in non-increasing order of weights.
$k \leftarrow$ First_k$(P, t, \mathbf{w}, m)$
$a_j \leftarrow 1$ for all $j \in [1 : k]$ ▷ Assign players to 1st resource
$x \leftarrow 2$ ▷ Define resource counter
$i \leftarrow k + 1$ ▷ Define player counter
$t \leftarrow W[1 : k]$ ▷ Update load threshold
while $x \leq m$ and $i \leq n$ **do**
  Compute relevant input $\mathcal{I}$
  $k \leftarrow$ Subsequent_k$(P, t, \mathbf{w}[i : n], m - x + 1, \mathcal{I})$
  $a_j \leftarrow x$ for all $j \in [i, i + k]$ ▷ Assign players to resource x
  $x \leftarrow x + 1$ ▷ Update resource counter
  $t \leftarrow W[i : i + k - 1]$ ▷ Update load threshold
  $i \leftarrow i + k$ ▷ Update player counter
if $i < n$ **then** ▷ Some players remain unassigned
  return None
else ▷ All players assigned to resources
  $a \leftarrow (a_1, ..., a_n)$
  if Satisfies$(\mathbf{a}, P)$ **then** ▷ Test $\mathbf{a}$ for P
    return $a = (a_1, ..., a_n)$
  else
    return None

3.1 Simple Satisfiability

For a number of simple properties, one can always find a P-assignment (rendering the decision problem P-Satisfiability trivial) in polynomial time, while P-Makespan is NP-complete. We now discuss such properties P, and address the corresponding P-Makespan problems in Section 4.

$\top$-Satisfiability. Any assignment satisfies $\top$ when there is no makespan threshold, and thus its existence is guaranteed, but not uniqueness.

Eq-Satisfiability. Consider an assignment where all players are assigned to the same resource. This satisfies *Eq* since all players incur the same cost W (which is the load of the resource to which they are assigned). Its existence is thus guaranteed, but not uniqueness.

Cr-Satisfiability. The Longest Processing Time algorithm [10] is known to find a *Cr*-assignment in time $O(n \log n)$ [8]. It is a greedy algorithm that first sorts players by non-increasing weight, and then sequentially assigns them to the resource with the least current load. This result is included in Table 1.

3.2 Envy-Freeness

Envy-freeness as a fairness property is extensively discussed in [13, 14] and not the primary focus of this paper. For an algorithm that solves *EF*-Makespan and $EF \wedge Cr$-Makespan in time $O(n\sqrt{n})$, see the extended version of our paper [15].

3.3 Weak Ordered Envy-Freeness

Since *WOE* does not place any restrictions on players assigned to the same resource (i.e., a player cannot be envious of another that is assigned to the same resource), assigning all players to a single resource satisfies *WOE*. The existence of a *WOE*-assignment is, therefore, guaranteed, and it can be returned in time $O(n)$. However, it may not be unique. For example, the assignments $\mathbf{a}$ and $\mathbf{a}'$ where $l(\mathbf{a}) = \{\{2, 2\}, \{2, 1\}\}$ and $l(\mathbf{a}') = \{\{2, 2, 2\}, \{1\}\}$ both satisfy *WOE*. Moreover, we can make a stronger structural statement about *WOE*-assignments.

Lemma 5. *Every WOE-assignment is contiguous.*

Hence, it is sufficient to consider only contiguous assignments to answer P-Makespan if P implies *WOE*. In addition, we derive a lemma saying that under the canonical ordering, if *WOE* holds between players in adjacent resources, then *WOE* holds globally.

Lemma 6. *Let* $\mathbf{a}$ *be a contiguous assignment in its canonical ordering. If the WOE condition holds between players assigned to any two adjacent resources, then* $\mathbf{a}$ *satisfies WOE.*

Lemma 5 suggests that Algorithm 1 can be used to solve P-Makespan whenever *WOE* is entailed by P. Hence, we obtain a correct algorithm by specifying the subroutines. Note that the proof uses Lemma 6 to prove *WOE*.

Theorem 7. *P-Makespan can be solved in time $O(n \log n)$ for $P \in \{WOE, WOE \wedge Cr, WOE \wedge Eq \wedge Cr\}$, and in time $O(n^2)$ for $P = WOE \wedge Eq$.*

While we defer the proof of Theorem 7 to the extended version of our paper [15], we subsequently discuss the specific subroutines for Algorithm 1 and their intuition.

3.3.1 WOE-Makespan. We specify the three subroutines called by Algorithm 1 to solve the *WOE*-Makespan. The idea is to assign as many players (in non-increasing weight order) to a resource as possible. This reduces the propensity for a lighter player that is yet to be assigned to envy a heavier player that is assigned to a high-load resource, and minimizes the total number of resources required to assign all players.

procedure First_k($P = WOE, t, \mathbf{w}, m$)
  return $\max\{i \in [n] : W[1:i] \leq t\}$

This subroutine assigns the maximum number of players to the first resource without exceeding the makespan threshold t. Notice that for *WOE*-Satisfiability where $t = \infty$, the subroutine returns $k = n$, i.e., all players are assigned to the first resource.

For the assignment of players to subsequent resources, let $\mathcal{I}$ be the largest weight assigned to the previous resource. This information is used to determine the largest number of players that can be assigned to the next resource without triggering envy, i.e., we need to ensure that the load on the current resource minus its lightest player does not exceed the load on the previous resource minus its heaviest player. The min operation when we return ensures that we do not exceed the number of remaining players.

procedure Subsequent_k($P = WOE, t, \mathbf{w}, m, \mathcal{I}$)
  Set $\hat{i} = \max\{i : W[1:i] \leq t - \mathcal{I}\}$
  return $\min(\hat{i} + 1, |\mathbf{w}|)$

By construction, *WOE* is satisfied between players assigned to any two adjacent resources. Hence, by Lemma 6, the candidate assignment satisfies *WOE*. So no further check for *WOE*-satisfaction is required and the Satisfies subroutine always returns True.

3.3.2 WOE∧Eq-Makespan. We begin with a structural proposition about $WOE \wedge Eq$-assignments, which says that if players of the same weight are assigned to different resources, then all players sharing those resources must be of the same weight.

Lemma 8. *Let* $\mathbf{a}$ *be a $WOE \wedge Eq$-assignment. Assume that $i, j \in [n]$ with $w_i = w_j$ and $a_i \neq a_j$. Then $w_k = w_j$ for all $k \in [n]$ with $a_k = a_i$ or $a_k = a_j$.*

We thus need to modify our previous *WOE* subroutines to ensure *Eq*-satisfaction by changing the way k is determined in First_k and Subsequent_k:

First, if not all players assigned to the same resource have the same weight, then the next player to be assigned to the next resource (the $(k+1)$st player) cannot share the same weight as the last player of the current assignment (the kth player). Second, if all players assigned to the same resource have the same weight, then all players of such weight must be evenly distributed among some number of resources so they incur the same cost. I.e., if k players of the same weight w_i are assigned to some resource, then it must be the case that $k \mid \#(w_i)$ and all assigned resources have the same load.

The subroutine First_k for $WOE \wedge Eq$-Makespan is as follows.

procedure First_k($P = WOE \wedge Eq, t, \mathbf{w}, m$)
  Set $n = |\mathbf{w}|$, $w_{n+1} = 0$
  if $\{i \in [n] : (W[1:i] \leq t) \wedge (w_{i+1} \neq w_i)\} \neq \emptyset$ **then**
    return $\max\{i \in [n] : (W[1:i] \leq t) \wedge (w_{i+1} \neq w_i)\}$
  else
    return $\max\{i \in [n] : (i \mid \#(w_1)) \wedge (i \cdot w_1 \leq t)\}$

For Subsequent_k, we need a number of additional information to ensure satisfaction of $WOE \wedge Eq$. Let $\mathcal{I}$ be the tuple $(\underline{w}, \overline{w}, k')$, representing the weight of the last player, the weight of the first player, and the number of players assigned to the previous resource, respectively. $\underline{w}$ and k' are used to ensure satisfaction of Eq by checking whether the current first player shares the same weight as the last player assigned to the previous resource; if so, then we must assign k' players (of the same weight) to the current resource. $\overline{w}$ is used to ensure satisfaction of WOE as for $P = WOE$.

procedure Subsequent_k($P = WOE \wedge Eq, t, \mathbf{w}, m, \mathcal{I}$)
  Let $\mathcal{I} = (\overline{w}, \underline{w}, k')$
  if $w_1 = \underline{w}$ **then** ▷ Continuation of players of equal weight
    return k'
  Set $n = |\mathbf{w}|$, $b = t - \overline{w}$, $w_{n+1} = 0$.
  if $\{i \in [n] \colon (W[1:i-1] \leq b) \wedge (w_{i+1} \neq w_i)\} \neq \emptyset$ **then**
    return $\max\{i \in [n] \colon (W[1:i-1] \leq b) \wedge (w_{i+1} \neq w_i)\}$
  else
    return $\max\{i \in [n] \colon (i \mid \#(w_1)) \wedge ((i-1) \cdot w_1 \leq b)\}$

Again, no further checks for whether $WOE \wedge Eq$ is satisfied is required, so the Satisfies subroutine always returns True.

3.3.3 $WOE \wedge Cr$-Makespan. While the WOE condition implicitly requires that resources assigned later have no greater load than the resources before it, the Cr condition requires that such differences are not too large, avoiding that players assigned to earlier resources cannot benefit from deviating to later resources. The idea is that, in each iteration, the subroutines now assign a number of players k to a resource to ensure that the average load of the remaining resources, $\frac{W - W[1:k]}{m-1}$, can satisfy $WOE \wedge Cr$, i.e.,

$$W[1:k-1] \leq \frac{W - W[1:k]}{m-1} \leq W[1:k] \tag{1}$$

The first inequality is a necessary (but not sufficient) condition to satisfy Cr: the average load on the remaining resources must be sufficiently large to prevent player k from deviating to the least-loaded resource. We need only satisfy this for player k (and not others on their resource) to produce the largest lower bound.

The second inequality is a necessary (but not sufficient) condition to satisfy WOE: it ensures that the average load on the remaining resources is sufficiently small to ensure a canonical ordering of contiguous assignment (Lemma 5). It is easy to see that at most one value of k satisfies these bounds.

Lemma 9. *At most one value of k satisfies Equation* (1).

Hence, we can apply the bounds in every iteration to reach a unique canonical ordering of resources if one exists.

Corollary 10 (Uniqueness)**.** *If* $\mathbf{a}$ *and* $\mathbf{a}'$ *satisfy $WOE \wedge Cr$, then* $\mathbf{a}$ *and* $\mathbf{a}'$ *are equivalent.*

The example below demonstrates that k may not exist.

Example 11. *Consider the instance where* $\mathbf{w} = (3, 2, 2)$ *and* $m = 2$. *If $k = 1$, then we get $(W - W[1\colon 1])/(m-1) = 4$, which exceeds the upper-bound $W[1\colon 1] = 3$. If $k = 2$, then $(W - W[1\colon 2])/(m-1) = 2$, which falls below the lower-bound $W[1\colon 1] = 3$.*

More explicitly, the assignment of players to resources is governed by the following subroutines.

procedure First_k($P = WOE \wedge Cr, t, \mathbf{w}, m$)
  if $\exists k \colon W[1:k-1] \leq \frac{W - W[1:k]}{m-1} \leq W[1:k] \leq t$ **then**
    return k
  else return None

procedure Subsequent_k($P = WOE \wedge Cr, t, \mathbf{w}, m, \mathcal{I}$)
  if $m = 1$ **then** ▷ Final resource
    return $|\mathbf{w}|$
  if $\exists k \colon W[1:k-1] \leq \frac{W - W[1:k]}{m-1} \leq W[1:k] \leq t$ **then**
    return k
  else return None

Finally, even if the iterations of assigning players to resources result in a unique candidate assignment $\mathbf{a}$, $\mathbf{a}$ has only satisfied the necessary, but not sufficient, conditions of $WOE \wedge Cr$. Thus a final test is required to check that the candidate assignment satisfies $WOE \wedge Cr$. For WOE, we need only check that the lightest player of each resource assignment does not envy the heaviest player assigned to the previous resource, and there are at most $m - 1 < n$ cases to check; for Cr, since resource loads are ordered non-increasingly, we need only check that the heaviest player of each resource does not benefit from deviating to the last and least-loaded resource, for which there are again $m - 1 < n$ cases to check.

3.3.4 $WOE \wedge Eq \wedge Cr$-Makespan. Since the previous algorithm for $WOE \wedge Cr$-Makespan produces only a single feasible candidate, we need only conduct an additional check on whether this candidate satisfies Eq in the Satisfies subroutine to answer this case. The subroutines First_k and Subsequent_k remain unchanged.

3.3.5 Satisfiability Problems. Notice that for WOE-Satisfiability and $WOE \wedge Eq$-Satisfiability, assigning all players to the first resource is always a solution, which can be returned in time $O(n)$. Also, because there is at most one feasible assignment for the $WOE \wedge Cr$-Makespan and $WOE \wedge Eq \wedge Cr$-Makespan problems, a solution is unique if it exists. Theorem 7 implies that the corresponding satisfaction problem (i.e., with $t = \infty$) can be solved in $O(n \log n)$.

Corollary 12. *WOE-Satisfiability and $WOE \wedge Eq$-Satisfiability can be solved in time $O(n)$. $WOE \wedge Cr$-Satisfiability and $WOE \wedge Eq \wedge Cr$-Satisfiability can be solved in time $O(n \log n)$.*

3.4 Strong Monotonicity

We first notice that an SM-assignment is always contiguous: if it was not, then under any non-increasing ordering of players and resources we can find some $i, j \in [n] : w_i < w_j$ and $a_i < a_j$, which contradicts SM. Another observation we can make is that in an SM-assignment, all players sharing the same resource must have the same weight; otherwise these players would incur the same cost, which fails the SM condition for the lighter players.

Lemma 13. *Let* $\mathbf{a}$ *be an SM-assignment. Then* $\mathbf{a}$ *is contiguous. Moreover, if $a_i = a_j$ for players $i, j \in [n]$, then $w_i = w_j$.*

In the remainder of this section, we discuss and present the relevant subroutines of the SCA for each property containing SM, to support the following theorem.

Theorem 14. *P-Makespan and P-Satisfiability can be solved in time $O(n \log n)$ for $P \in \{SM, SM \wedge Eq, SM \wedge Cr, SM \wedge Eq \wedge Cr\}$.*

3.4.1 SM-Makespan. By Lemma 13, a natural way of searching for an *SM*-assignment is to look for its canonical ordering, and sequentially assign all players of a given weight to as few resources as possible while ensuring *SM*-satisfaction.

In First_k, if $\#(w_1)$ is too large for all players of weight w_1 to be assigned to the first resource, then we first calculate the minimum number of resources required to accommodate all $\#(w_1)$ players, then distribute these players as evenly as possible, to ensure that the last resource to which players of weight w_1 are assigned has the highest possible load. We do this to ensure that the new upper bound on the load of the next resource is as high as possible.

procedure First_k($P = SM, t, \mathbf{w}, m$)
  if $\#(w_1) \cdot w_1 \leq t$ **then**
    return $\#(w_1)$
  else
    Compute $\hat{k} = \max\{k : k \cdot w_1 \leq t\}$
    Set $r = \left\lceil \frac{\#(w_1)}{\hat{k}} \right\rceil$
    return $\lceil \#(w_1)/r \rceil$

We use a similar process for Subsequent_k, except that an additional check occurs on whether the current first player has the same weight as the players assigned to the previous resource. If so, then the load threshold is a weak inequality, since players of the same weight can incur the same cost. If not, then the load threshold is a strict inequality, to ensure that the players of the lighter weight incur strictly smaller cost than the heavier players.

procedure Subsequent_k($P = SM, t, \mathbf{w}, m, \mathcal{I}$)
  Set $\mathcal{I}$ to be the weight of players in previous resource
  if $\mathcal{I} = w_1$ **then** ▷ Load must be $\leq t$
    if $\#(w_1) \cdot w_1 \leq t$ **then**
      return $\#(w_1)$
    else
      Compute $\hat{k} = \max\{k : k \cdot w_1 \leq t\}$
      Set $r = \left\lceil \frac{\#(w_1)}{\hat{k}} \right\rceil$
      return $\lceil \#(w_1)/r \rceil$
  else ▷ Load must be $< t$
    if $\#(w_1) \cdot w_1 < t$ **then**
      return $\#(w_1)$
    else
      Compute $\hat{k} = \max\{k : k \cdot w_1 < t\}$
      Set $r = \left\lceil \frac{\#(w_1)}{\hat{k}} \right\rceil$
      return $\lceil \#(w_1)/r \rceil$

Finally, the subroutines above ensure that there are no additional checks to be made, so the Satisfies subroutine simply returns true.

3.4.2 SM∧Eq-Makespan. For a solution to *SM*∧*Eq*-Makespan, we need to additionally ensure that all players of the same weight incur exactly the same cost. This means that they must be distributed evenly across a number of resources (rather than the minimum number of resources), as shown in the following subroutines.

procedure First_k($P = SM \wedge Eq, t, \mathbf{w}, m$)
  if $\#(w_1) \cdot w_1 \leq t$ **then**
    return $\#(w_1)$
  else
    return $\max\{k : (k \cdot w_1 \leq t) \wedge (k \mid \#(w_1))\}$

procedure Subsequent_k($P = SM \wedge Eq, t, \mathbf{w}, m, \mathcal{I}$)
  Let $\mathcal{I}$ be the weight of players in previous resource
  if $\mathcal{I} = w_1$ **then** ▷ Load must be $\leq t$
    if $\#(w_1) \cdot w_1 \leq t$ **then**
      return $\#(w_1)$
    else
      return $\max\{k : (k \cdot w_1 \leq t) \wedge (k \mid \#(w_i))\}$
  else ▷ Load must be $< t$
    if $\#(w_1) \cdot w_1 < t$ **then**
      return $\#(w_1)$
    else
      return $\max\{k : (k \cdot w_1 < t) \wedge (k \mid \#(w_i))\}$

Again, no further checks are required, so the Satisfies subroutine always returns True.

3.4.3 SM ∧ Cr-Makespan. To satisfy *Cr* we cannot simply assign the maximal number of players to each resource, as doing so might result in a resource being unused or having a low load, and thus make it beneficial to deviate to this resource. We need to ensure that when choosing the number of players k to assign to each resource, the remaining players can be distributed so that all subsequent resources have sufficiently high load.

To ensure that an assignment satisfies *Cr*, we can impose the bounds in Equation (1) at each iteration of the assignment. As we have shown in Corollary 10, at most one value of k exists at each iteration, so there is a unique candidate solution which we can find and check for the *SM* ∧ *Cr*-Makespan problem.

Since players sharing the same resource must be of the same weight, we can simplify Equation (1) as $(k-1)w_1 \leq \frac{W-kw_1}{m-1} \leq kw_1$, where k is the number of players of weight w_1 assigned to the resource. The only integer value that k can take is $\lceil \frac{W}{mw_1} \rceil$, if it exists, and we can check if it is less than or equal to $\#(w_1)$, the number of remaining players of weight w_1.[2]

Repeating this step for each resource eventually either yields that there is no *SM* ∧ *Cr*-assignment, or that all players are assigned, at which point we can check whether the candidate assignment satisfies *Cr* overall. This can be done by checking whether each player has the incentive to deviate to the least-loaded resource.[3] We specify the following subroutines to Algorithm 1.

procedure First_k($P = SM \wedge Cr, t, \mathbf{w}, m$)
  if $\exists k : (k-1)w_1 \leq \frac{W-kw_1}{m-1} \leq kw_1 \leq t \wedge k \leq \#(w_1)$ **then**
    return k
  else return None

procedure Subsequent_k($P = SM \wedge Cr, t, \mathbf{w}, m, \mathcal{I}$)
  Let $\mathcal{I}$ be the weight of players in previous resource
  if $\mathcal{I} = w_1$ **then** ▷ Load must be $\leq t$
    if $m = 1$ **then** ▷ Final resource
      if $w_1 = w_{|\mathbf{w}|}$ **then**
        return $|\mathbf{w}|$
      else return None
    if $\exists k : (k-1)w_i \leq \frac{W-kw_i}{m-1} \leq kw_i \leq t \wedge k \leq \#(w_1)$ **then**
      return k
    else return None

[2] Rearranging the bounds gives $\frac{W}{mw_1} \leq k \leq \frac{W}{mw_1} + \frac{m-1}{m}$.

[3] It is in fact sufficient to check that the heaviest player assigned to each resource has no incentive to deviate to the least-loaded resource.

```
    else                                              ▷ Load must be < t
        if m = 1 then                                 ▷ Final resource
            if w_1 = w_{|w|} ∧ W[1 : |w|] < t then
                return |w|
            else return None
        if ∃k : (k − 1)w_1 ≤ (W − k w_1)/(m − 1) ≤ k w_1 < t ∧ k ≤ #(w_1)
    then
            return k
        return None

procedure Satisfies(a, P = SM ∧ Cr)
    for i ∈ [n] do
        if v_{a_i} > v_m + w_i then                   ▷ Fails Cr
            return None
    return True
```

3.4.4 SM ∧ Eq ∧ Cr-Makespan. Since the algorithm for *SM* ∧ *Cr*-Makespan produces a single feasible candidate solution (if it exists), we simply need to additionally check for *Eq* in the final part of the algorithm. This can be done in $O(n)$ by checking whether all players with the same weight incur the same cost.

3.4.5 Satisfiability Problems. There is no existence guarantee for *P*-Satisfiability for properties that entail *SM*. For example, the instance given in Figure 2 does not have an *SM*-assignment. Furthermore, *SM*- and *SM* ∧ *Eq*-assignments need not be unique. Consider the load distribution of two assignments for the instance $\mathbf{w} = (5, 3, 1)$, $w = 2$: $\{\{4\}, \{2, 1\}\}$ and $\{\{4, 2\}, \{1\}\}$. They are distinct assignments and both satisfy *SM* ∧ *Eq*. Finally, as discussed in Section 3.4.3, *SM* ∧ *Cr*- and *SM* ∧ *Eq* ∧ *Cr*-assignments are unique if they exist.

3.5 *WOE* ∧ *SM*

For completeness, we briefly discuss properties containing *WOE* ∧ *SM*. For *WOE* ∧ *SM*, we can base our subroutines on those for *SM*, with the added requirement of no weak ordered envy between players in consecutive resources. A similar argument applies for *WOE* ∧ *SM* ∧ *Eq*. For *WOE* ∧ *SM* ∧ *Cr* and *WOE* ∧ *SM* ∧ *Eq* ∧ *Cr*, we can use the corresponding First_k and Subsequent_k of either the *WOE* version or the *SM* version, and simply check for satisfaction of the other condition in the subroutine Satisfies. This is sufficient due to the uniqueness of a satisfying assignment, if one exists. In all cases, a solution is found in time $O(n \log n)$. For brevity, we omit these results from Table 1.

3.6 Weak Monotonicity

P-Makespan when *P* implies *WM* is generally not tractable, except for the special cases of *WM*-Satisfiability and *WM* ∧ *Eq*-Satisfiability, where the simple assignment of placing all players on the same resource trivially satisfies *WM* ∧ *Eq*. Discussion on the intractability of *WM*-Makespan is contained in the next section.

4 INTRACTABLE PROPERTIES

Finally, we discuss properties *P* for which *P*-Makespan is intractable. In contrast to the previous section, these problems are more difficult because a satisfying assignment of the lowest makespan may not be contiguous, as the example below shows for the case of *WM*.

Example 15. *Consider the instance with* $\mathbf{w} = (4, 3, 2, 1)$ *and* $m = 2$. *The assignment* $\mathbf{a}$ *given by* $l(\mathbf{a}) = \{\{4, 1\}, \{3, 2\}\}$ *is of the lowest makespan of* 5, *and satisfies WM since all players incur the same cost. It is, however, not contiguous. The makespan-optimal, contiguous WM-assignment has load distribution* $\{\{4, 3\}, \{2, 1\}\}$ *and a makespan of* 7.

Our reductions are from the known NP-hard decision problem, Partition [11]. An instance is given by a vector $\mathbf{s} = (s_1, ..., s_n)$ of positive integers. It is a Yes-instance if there exists a subset $S \subset [n]$ such that $\sum_{i \in S} s_i = \sum_{i \in [n] \setminus S} s_i$.

The key of our reduction is to set $S = \sum_i s_i$, and to define the reduced instance by $\mathbf{w} = (\mathbf{s}, S + 1, S + 1) \in \mathbb{Z}_{>0}^{n+2}$ and $m = 2$. For makespan minimization, we additionally set $t = \frac{3}{2}S + 1$. Note that hardness for satisfiability of *P* immediately implies hardness of makespan minimization (by setting *t* to be sufficiently large) but our second result also covers properties for which we can efficiently solve *P*-Satisfiability.

Theorem 16. *Deciding whether a P-assignment exists is NP-complete for any property P in* $\{Eq \wedge Cr, WM \wedge Cr, WM \wedge Eq \wedge Cr\}$.

Theorem 17. *Deciding whether a P-assignment satisfying a given makespan threshold exists is NP-complete for any property P in* $\{\top, Eq, Cr, Eq \wedge Cr, WM, WM \wedge Eq, WM \wedge Cr, WM \wedge Eq \wedge Cr\}$.

Note that an assignment need not be unique if it exists. For example, the two assignments $\mathbf{a}$ and $\mathbf{a}'$ where $l(\mathbf{a}) = \{\{2, 1, 1\}, \{2, 1, 1\}\}$ and $l(\mathbf{a}') = \{\{2, 2\}, \{1, 1, 1, 1\}\}$ are clearly not equivalent, and both satisfy $WM \wedge Eq \wedge Cr$.

Finally, we reflect on why the same reduction for NP-hardness does not work for properties containing *WOE*, *SM*, or *EF*. These properties are not satisfied whenever players of different weights (say $w_i < w_j$) are assigned to different resources $a_i \neq a_j$ of the same load $v_{a_i} = v_{a_j}$, because this implies that $v_{a_i} \not< v_{a_j}$ (which fails *SM*) and $v_{a_i} - w_i > v_{a_j} - w_j$ (which fails *WOE* and *EF*). A Yes-instance of Partition would, therefore, not result in a Yes-instance of *P*-Makespan under the same reduction.

5 CONCLUSION

In the context of equilibrium refinement, the problem of finding a 'fair' Nash equilibrium can be phrased as solving the *P*-Satisfiability problem where *P* implies *Cr* and some additional fairness properties. This problem turned out to be tractable for a large number of fairness classes. Even beyond equilibria, this paper shows that the intractability of scheduling on identical machines is lifted when imposing additional, desirable constraints. The reason is that these constraints impose important structure to the solutions (i.e., contiguity), which sufficiently reduces the search space.

Many research questions follow naturally from our results. What is the price of anarchy for the fair coordination problem? For problems where the solution to *P*-Satisfiability is guaranteed to exist, how does its optimal solution compare to that of the unconstrained case? This gives a measure of the efficiency-fairness trade-off. In the context of communication partition, the tractable fairness conditions are useful for efficiently finding the optimal partition, by removing the combinatorial blow-up at the coalitional level. However, there may still exist combinatorial difficulty at the partition level, so further investigation is necessary.

ACKNOWLEDGMENTS

Most of this work was done when Martin Bullinger was at the University of Oxford. Wei-Chen Lee was kindly supported by the Oxford-Taiwan Graduate Scholarship and the Oxford-DeepMind Graduate Scholarship.

REFERENCES

[1] Sebastian Aland, Dominic Dumrauf, Martin Gairing, Burkhard Monien, and Florian Schoppmann. 2011. Exact Price of Anarchy for Polynomial Congestion Games. *SIAM J. Comput.* 40, 5 (Jan. 2011), 1211–1233. https://doi.org/10.1137/090748986

[2] Baruch Awerbuch, Yossi Azar, and Amir Epstein. 2005. The Price of Routing Unsplittable Flow. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC05)*. ACM. https://doi.org/10.1145/1060590.1060599

[3] Kshipra Bhawalkar, Martin Gairing, and Tim Roughgarden. 2014. Weighted Congestion Games: The Price of Anarchy, Universal Worst-Case Examples, and Tightness. *ACM Transactions on Economics and Computation* 2, 4 (Oct. 2014), 1–23. https://doi.org/10.1145/2629666

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

[5] George Christodoulou and Elias Koutsoupias. 2005. *On the Price of Anarchy and Stability of Correlated Equilibria of Linear Congestion Games, ,.* Springer Berlin Heidelberg, 59–70. https://doi.org/10.1007/11561071_8

[6] George Christodoulou and Elias Koutsoupias. 2005. The price of anarchy of finite congestion games. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing* (Baltimore, MD, USA). Association for Computing Machinery, New York, NY, USA, 67–73. https://doi.org/10.1145/1060590.1060600

[7] Duncan K. Foley. 1967. Resource allocation and the public sector. *Yale Economics Essays* 7 (1967), 45–98.

[8] Dimitris Fotakis, Spyros Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul Spirakis. 2002. The Structure and Complexity of Nash Equilibria for a Selfish Routing Game. In *Automata, Languages and Programming*, Peter Widmayer, Stephan Eidenbenz, Francisco Triguero, Rafael Morales, Ricardo Conejo, and Matthew Hennessy (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 123–134.

[9] Michael R. Garey and David S. Johnson. 1979. Computers and intractability: a guide to the theory of np-completeness. *WH Free. Co., San Fr* 38 (1979), 40.

[10] R. L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.* 17, 2 (1969), 416–429. http://www.jstor.org/stable/2099572

[11] Richard M. Karp. 1972. *Reducibility among Combinatorial Problems.* Springer US, Boston, MA, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[12] Elias Koutsoupias and Christos Papadimitriou. 2009. Worst-case equilibria. *Computer Science Review* 3, 2 (May 2009), 65–69. https://doi.org/10.1016/j.cosrev.2009.04.003

[13] Wei-Chen Lee, Alessandro Abate, and Michael J. Wooldridge. 2025. Equilibrium Selection via Cheap-talk Partition: Extended Abstract. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems* (2025).

[14] Wei-Chen Lee, Alessandro Abate, and Michael J. Wooldridge. 2025. Partition Equilibria in Weighted Singleton Congestion Games. *Proceedings of the European Conference on Artificial Intelligence* (2025).

[15] Wei-Chen Lee, Martin Bullinger, Alessandro Abate, and Michael Wooldridge. 2025. Fair Coordination in Strategic Scheduling. arXiv:2512.13244 [cs.GT] https://arxiv.org/abs/2512.13244

[16] Igal Milchtaich. 1996. Congestion Games with Player-Specific Payoff Functions. *Games and Economic Behavior* 13, 1 (1996), 111–124. https://doi.org/10.1006/game.1996.0027

[17] Dov Monderer and Lloyd S. Shapley. 1996. Potential Games. *Games and Economic Behavior* 14, 1 (May 1996), 124–143. https://doi.org/10.1006/game.1996.0044

[18] Robert W. Rosenthal. 1973. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory* 2, 1 (Dec. 1973), 65–67. https://doi.org/10.1007/bf01737559

[19] H. R. Varian. 1974. Equity, Envy, and Efficiency. *Journal of Economic Theory* 9 (1974), 63–91.

[20] Berthold Vöcking. 2007. *Selfish Load Balancing.* Cambridge University Press, 517–542.