

Online Scheduling via Gradient Descent for Weighted Flow Time Minimization*

Qingyun Chen[†]

Sungjin Im[‡]

Aditya Petety[§]

Abstract

In this paper, we explore how a natural generalization of Shortest Remaining Processing Time (SRPT) can be a powerful *meta-algorithm* for online scheduling. The meta-algorithm processes jobs to maximally reduce the objective of the corresponding offline scheduling problem of the remaining jobs: minimizing the total weighted completion time of them (the residual optimum). We show that it achieves scalability for minimizing total weighted flow time when the residual optimum exhibits *supermodularity*. Scalability here means it is $O(1)$ -competitive with an arbitrarily small speed augmentation advantage over the adversary, representing the best possible outcome achievable for various scheduling problems.

Thanks to this finding, our approach does not require the residual optimum to have a closed mathematical form. Consequently, we can obtain the schedule by solving a linear program, which makes our approach readily applicable to a rich body of applications. Furthermore, by establishing a novel connection to *substitute valuations in Walrasian markets*, we show how to achieve supermodularity, thereby obtaining scalable algorithms for various scheduling problems, such as matroid scheduling, generalized network flow, and generalized arbitrary speed-up curves, etc., and this is the *first* non-trivial or scalable algorithm for many of them.

1 Introduction

Scheduling is a fundamental algorithmic problem because it is at the heart of any resource allocation process. Because of its wide appearance in practice, scheduling has been a central field in many disciplines such as operations research and computer science. Further, scheduling problems have provided arenas where theoretical computer science has developed and tested new theory. Scheduling theory has also become richer from various points of view, such as approximation, online, stochastic, game theory, etc. In the modern era, scheduling is giving increasing challenges and opportunities as more powerful computing resources are becoming available at low cost for novel applications.

Online scheduling studies dynamic scheduling settings and is found in various applications. In the online setting, the algorithm must make scheduling decisions without knowing jobs arriving in the future. Competitive analysis is commonly used to measure the quality of an online algorithm in contrast to the optimal offline algorithm [BEY05]. Unfortunately, in competitive analysis, online algorithms are often too weak compared to their offline adversaries that know the whole input sequence from the beginning. Particularly, the effect of online scheduling algorithms making sub-optimal decisions can accrue over time, often rendering their competitive analysis pointless.

To overcome this pessimistic landscape, in the mid-'90s, a resource augmentation model was introduced to online scheduling [KP00, PSTW02]. For example, if the augmented resource is speed, which is the most commonly augmented resource, the scheduling algorithm is compared to the adversary that can process jobs slightly slower. This beyond-worst-case analysis enabled meaningful study of a large number of scheduling problems, which would otherwise not be possible [PST04]. New analysis tools have been developed for online scheduling, such as potential functions [IMP11] and dual fitting [AGK12, GKP12], which have resulted in a plethora of new exciting results for various online scheduling problems under the resource augmentation model.

*This work was supported in part by NSF grants CCF-1844939, CCF-2121745, and CCF-2423106, and by ONR grant N00014-22-1-2701.

[†]Electrical Engineering and Computer Science, University of California, 5200 N. Lake Road, Merced CA 95344. qchen41@ucmerced.edu.

[‡]Computer Science and Engineering, University of California, 1156 High Street, Santa Cruz, CA 95064. sungjin@ucsc.edu.

[§]Electrical Engineering and Computer Science, University of California, 5200 N. Lake Road, Merced CA 95344. apetety@ucmerced.edu.

Despite all this success, our understanding of online scheduling can be considerably improved. First of all, we only have limited ways to design online scheduling algorithms systematically. For example, the powerful online primal-dual framework [BSN09] is mainly applicable for monotone packing and covering problems but a large number of scheduling problems are not as such. Dual fitting tries to analyze a candidate algorithm by setting the primal and dual LP variables according to the algorithm's behavior. Thus, designing online scheduling algorithms still requires much creativity, often non-trivially combining commonly used scheduling strategies via repeated trial and error. There is a need to find new algorithm design strategies that can keep pace with increasingly complicated scheduling environments.

Up to date, Proportional Fairness (PF) has been the only online meta scheduling algorithm known. The algorithm seeks to maximize the product of how fast jobs get processed at each time (subject to the scheduling constraints), and it generalizes Round-Robin.¹ It was only recently shown that PF is $O(1/\epsilon)$ -competitive for a large class of scheduling problems, given $(e + \epsilon)$ -speed over the adversary [IKM18]. Specifically, it was shown that if we view jobs as agents and how fast they get processed as their utility, PF finds a fair resource allocation that leads to a competitive schedule for the total flow time objective. The algorithm was not new, but the main contribution was to demonstrate its effectiveness as a meta scheduling algorithm—the guarantee holds true for various problems as long as the allocation is monotone under PF.²

In this paper we aim to discover more meta scheduling algorithms that work for various scheduling environments. We focus on perhaps the most widely used metric, namely average (weighted) flow time, which measures the average time a job stays in the system from its arrival until completion.

Roadmap. Due to the general applicability of our approach, we will employ somewhat heavy notation. Further, we use concepts from market equilibria, which are not commonly used in the online scheduling/algorithm literature. To make the paper more accessible, we first describe our results informally (Section 1.1) and then illustrate them using a single machine scheduling example (Section 1.2). Readers unfamiliar with online scheduling are strongly recommended to **read Section 1.2 before Section 1.1**. There is some intentional overlap between these sections.

We then formally define our problem and introduce preliminary concepts about online scheduling and substitute valuations in Section 2. The linear substitute results and examples discussed in Section 2.4 are, to our best knowledge, new. We formally present our algorithms and theorems in Section 3. Section 4 is devoted to proving the main theorems. Section 5 demonstrates how these theorems are applied to the applications discussed in Section 1.3. Other related work is discussed in Section 6. Appendix C discusses solving the residual optimum problem efficiently and implementing our meta algorithm efficiently. All missing proofs are provided in the Appendix.

1.1 Informal Description of Our Results and Contributions We seek to find a meta scheduling algorithm that works for a large class of problems that fall into the polytope scheduling, which encompasses numerous scheduling problems [IKM18].

Polytope Scheduling. In the Polytope Scheduling Problem (PSP), we are given a set of constraints that form a downward-closed³ polytope \mathcal{P} (more generally a convex body), which constrains the feasible processing rate vector at each time: each job j can be processed at a rate of z_j if and only if $\mathbf{z} := \{z_j\}_j \in \mathcal{P}$. For example, we have $\mathcal{P} = \{\mathbf{z} \mid \sum_j z_j \leq 1 \text{ and } \mathbf{z} \geq 0\}$ in the single machine scheduling, which says jobs can be processed at a rate of up to 1 in total. We assume that \mathcal{P} remains fixed over time. In the online setting, each job j arrives at time r_j with (original) processing size p_j and weight w_j . Note that the online scheduler learns about job j , including p_j and w_j , only when it arrives. Preemption is allowed and the goal is to minimize total (weighted) flow time.

To develop a general approach to this problem, we get an inspiration from the Shortest Remaining Processing Time (SRPT): The SRPT algorithm, which prioritizes jobs with the shortest remaining time, is well-known for optimally minimizing the average (unweighted) flow time of jobs on a single machine. Interestingly,

¹In Round-Robin scheduling for a single machine, jobs are processed in turns after each unit of processing time. We assume that each unit of time is infinitesimally small. Under this assumption, the algorithm is equivalent to Processor Sharing, where all active jobs are processed at an equal rate at all times. In the single machine, PF solves $\max \prod_{j=1}^n z_j$ s.t. $\sum_j z_j \leq 1$ and $\mathbf{z} \geq 0$, thus yields the same assignment as Round-Robin.

²This means that when another job is added to the PF allocation, the share or speed allocated to each existing job decreases.

³If $\mathbf{0} \leq \mathbf{z}' \leq \mathbf{z}$ and $\mathbf{z} \in \mathcal{P}$, then $\mathbf{z}' \in \mathcal{P}$.

Problems	Previous Results	Our Results
Matroid Scheduling	$e + \epsilon$ [IKM18]	$1 + \epsilon$
Generalized Flow Scheduling	unknown	$1 + \epsilon$
Unrelated Machines	$1 + \epsilon$ [AGK12]	$1 + \epsilon$
Speed-up Curves on Partitioned Resources	unknown	$1 + \epsilon$

Table 1: Comparison of our results to prior work that summarizes the speed needed to be $O(1_\epsilon)$ -competitive for minimizing total weighted flow time.

the scheduling literature implicitly suggests a possible interpretation of SRPT as a form of gradient descent [CGKM09, AGK12, BCP13, BPS09]. The residual optimum [CGKM09, AGK12], informally, is the min cost for completing all remaining jobs, assuming no more jobs arrive. Taking *gradient descent on the residual objective*, meaning that we process the jobs that *decrease the residual optimum the most*, is natural because we complete all the remaining jobs when the residual optimum becomes zero. Further, the spirit of gradient descent is present in the analysis based on potential functions [CGKM09, IMP11, EP12].

Inspired by this, we explore *gradient descent as a meta algorithm for online scheduling*. Our approach is general and streamlined: i) Algorithm Execution: Solve an LP relaxation of the offline residual optimum and perform gradient descent; and ii) Performance Guarantee: If a valuation function coming from the scheduling constraints is a substitute function, gradient descent is competitive.

Below, we outline our results and contributions.

1. **First Scalable Meta-algorithm.** We demonstrate the effectiveness of gradient descent as a *meta-algorithm*. We use gradient descent to give the *first scalable algorithm* for various scheduling problems, such as matroid scheduling, generalized flow scheduling, generalized arbitrary speed-up curves, etc.; see Section 1.3 for the problem definition. An algorithm is called scalable if it is $O(1_\epsilon)$ -competitive with $(1 + \epsilon)$ factor more speed given over the adversary. Scalable algorithms are essentially the best one can hope for problems that have strong lower bounds on competitive ratio, and all the above problems are as such. Since the aforementioned PF (Proportional Fairness) requires at least $(2 + \epsilon)$ -speed to be $O(1_\epsilon)$ -competitive [Edm99, IKM18], our result gives the first scalable meta algorithm. We also note that we give the *first non-trivial results* for generalized flow scheduling and generalized arbitrary speed-up curves. For a comparison of our results to prior work, please see Table 1.

As discussed earlier, the concept of gradient descent is implicitly embedded within the online scheduling literature. In this paper, we present a general approach to using gradient descent and establish a sufficient condition that ensures its competitiveness. Rather than showcasing gradient descent's applicability to several scheduling problems by case-by-case analysis, we seek a general approach that can be used to solve a wide range of scheduling problems.

2. **Reduction to Obtaining an Offline (LP) Relaxation with Supermodularity.** We *reduce* designing online scheduling algorithms to *showing supermodularity of the offline residual optimum problem*. The residual optimum is the minimum weighted completion time objective to complete the remaining jobs, assuming no more jobs arrive.
 - (a) The (approximate) residual optimum need not have a closed-form mathematical expression. Instead, it can be computed using mathematical programming, such as time-indexed linear programming (LP). This allows us to directly derive a gradient descent-based schedule. Consequently, our approach has a wide range of potential applications. In contrast, traditional potential function analysis relies on finding closed-form expressions, which has been a major hurdle in designing and analyzing online scheduling algorithms [IMP11]. Additionally, dual fitting is primarily an analysis tool, rather than a method for algorithm design.
 - (b) We show that *supermodularity* of the residual optimum is a sufficient condition for gradient descent to be scalable. Intuitively, supermodularity implies adding a new job can only increase the residual optimum more when there are more existing jobs. Towards this end, we generalize the beautiful potential

function introduced by [CGKM09] and crystallize the analysis by identifying supermodularity as the key to the analysis.

However, it is in general challenging to show the residual optimum is supermodular. We address this issue by showing that it suffices to check the constraints that characterize the scheduling problem for a large class of problems. For example, for scheduling m identical parallel machines (we can process up to m jobs at a rate of one at each time), we only need to take a close look at the valuation function $v(\mathbf{x}) = \max \sum_{0 \leq \mathbf{y} \leq \mathbf{x}} \mathbf{a} \cdot \mathbf{y}$ s.t. $\|\mathbf{y}\|_1 \leq m$ and $\|\mathbf{y}\|_\infty \leq 1$ for some vector $\mathbf{a} \geq 0$. Intuitively, the valuation function means this: Given a set of jobs J' , what is the maximum value we can get by choosing m jobs from J' when each job j has a value of a_j ? Obviously, it is obtained by choosing m jobs with the highest value of a_j . Alternatively, one can view the valuation as the weighted rank function of a uniform matroid of rank m .

3. **Substitutes Empower Gradient Descent.** We show that if maximizing a linear function subject to the constraints is a substitute function, then the residual optimum is supermodular. A valuation function is called *gross/linear substitutes* (GS/LS) [KC82, GS99, LLN06, MS09] if increasing a resource's price does not reduce the demand for other resources. Consider the above valuation function $v(\mathbf{x})$ for m identical machines: Items are individually priced at \mathbf{q} and valued at \mathbf{a} by you. If you buy m items (say $1, 2, \dots, m$) with the maximum value of $a_j - q_j$ (so, your utility is the valuation of the items you buy minus the payment for that), increasing item 1's price might make you buy a different item instead, but you would still want items $2, 3, \dots, m$. This shows that $v(\mathbf{x})$ is a gross substitute; linear substitute is a continuous extension of this concept.

Substitute valuations play a crucial role in understanding market equilibria in Walrasian markets where agents aim to maximize their utility, defined as the difference between their valuation of purchased items and the price paid. Notably, Walrasian markets admit an equilibrium when all agents have GS valuations. An equilibrium is an allocation of items to agents alongside with a price vector where every agent gets an allocation maximizing their utility and all items are allocated. Furthermore, LP duality implies that this equilibrium maximizes the total valuation of all agents, also known as social surplus.

We view the residual optimum problem (with a sign flip) as the social surplus maximization problem in Walrasian markets. Conceptually, we introduce a phantom agent for each time slot, where each job's unit processing is a distinct item. If the valuation function is gross/linear substitutes (GS/LS), we prove that the residual optimum is supermodular. This leverages the fact that GS/LS valuations are a subclass of submodular functions and closed under max-convolution [LLN06, MS09]. This sufficient condition simplifies supermodularity checks. For example, in matroid scheduling, showing the valuation function (essentially the weighted rank) is GS/LS is easier than proving supermodularity for the entire residual optimum.

Many scheduling scenarios involve jobs with processing rates dependent on allocated resources. To address this heterogeneity, linear substitutes (LS) [MS09] become more relevant. Linear substitutes provide a natural continuous extension of GS. Less studied than GS, LS exhibits distinct behavior. We present novel results for LS, including Theorem 2.15, which shows a concave closure of a GS valuation function is LS and Theorem 2.16, which shows generalized flow being LS. These findings hold independent interest. See Sections 2.3 and 2.4 for more details and formal definitions.

Our work builds upon and extends numerous groundbreaking results that have been established throughout the history of scheduling research, e.g. [LAHW97, BPS09, CGKM09, EP12, AGK12, GKP12, BCP13]. As a consequence, we demonstrate the efficacy of gradient descent as a meta-algorithm, which has perhaps been considered a possibility by many. This work perhaps suggests that SRPT, a common topic covered in algorithms and operating systems courses, should be taught alongside its connection to gradient descent.

1.2 Overview of Our Approach We illustrate our approach and its benefits using the single machine scheduling problem as a running example.

Single machine scheduling. A set of n jobs arrive over time to be processed on a single machine/processor and the scheduler is not aware of jobs that have not arrived yet. Job j has a processing time (size) p_j and arrives at time r_j . The scheduler is clairvoyant, meaning that it learns p_j upon j 's arrival. The single machine can process at most one job or equivalently one unit of jobs at any point in time. Thus, letting z_j denote the rate we process

job j at a time, feasible schedules are constrained by $\mathbf{z} \in \mathcal{P} = \{\mathbf{z} \geq 0 \mid \sum_j z_j \leq 1\}$.⁴ A job j is completed when it gets processed for p_j units of time in total. Preemption is allowed, meaning that the scheduler can preempt a job being processed and resume it from where it left off. If we denote j 's completion time as C_j , its flow time⁵ is defined as $F_j := C_j - r_j$. Suppose the objective is to find an online schedule that minimizes total (or equivalently average) flow time, i.e., $\sum_j F_j$.

For this simple problem, as mentioned before, at each time t , SRPT (Shortest Remaining Processing Time) processes the job j with the minimum $p_j(t)$, which denotes j 's remaining size at time t , that is, p_j minus the amount of time for which it has been processed until time t . SRPT optimally minimizes the total flow time—intuitively, it minimizes the overall delay of job processing by processing the job it can complete the earliest.

1.2.1 Supermodularity Makes Gradient Descent Work (Theorem 3.2) Let us first interpret SRPT as a form of gradient descent (GD). Suppose the current time is t . Since we would like to minimize the total flow time, $\sum_j (C_j - r_j)$, by ignoring the cost of each job j already incurred (how long it has waited), $t - r_j$, pretending the current time is 0, and assuming no more jobs arrive in the future, let us try to measure the minimum cost we should pay, which is termed the *residual* optimum. The residual optimum then becomes the minimum total completion time of jobs with sizes $\{p_j(t)\}_j$:

$$f(\mathbf{p}(t)) = np_{\pi(1)}(t) + (n-1)p_{\pi(2)}(t) + (n-2)p_{\pi(3)}(t) + \cdots + 1p_{\pi(n)}(t), \quad (1.1)$$

where jobs are ordered so $\pi(1)$ has the smallest remaining size and $\pi(n)$ has the largest; we hide π 's dependency on t for notational convenience. Note that computing $f(\mathbf{p}(t))$ is a purely *offline* problem. In general, $f(\mathbf{x})$ does not depend on a specific algorithm but only on the scheduling problem specified by the constraints, \mathcal{P} . Given the constraint that $\sum_j p_j(t)$ can be reduced by one at a time, GD processes job $\pi(1)$, because it decreases $f(\mathbf{p}(t))$ the most. This *coincides with SRPT*. Thus, GD decreases the residual objective by exactly n , the number of jobs currently alive, at a time. Note that the algorithm designer *only needs to know the residual objective to derive GD*. While $f(\mathbf{p}(t))$ has a nice closed form in the single machine case, it is often NP-hard to compute exactly. In such cases, we can use linear or convex programming (Sections 1.2.2 and 3.2).

Let $\mathbf{p}^O(t)$ denote the remaining job sizes under the adversary's schedule. Let A denote GD algorithm (in this case, SRPT) and O the adversary. Similarly, $A(t)$ and $O(t)$ imply the set of jobs currently alive at time t in their schedule; we may drop t for notational brevity. For analysis, we use the following *generic* potential function $\Phi(t)$, which is directly derived from f in a *black-box* manner:

$$\Phi(t) := \frac{2}{\epsilon} \left(f(\mathbf{p}(t)) - \frac{1}{2} f(\mathbf{p}(t) \parallel \mathbf{p}^O(t)) \right),$$

where $\mathbf{p}^O(t) := \{p_j^O(t)\}_j$ denotes a vector of remaining job sizes in O 's schedule and \parallel denotes concatenation of two vectors. In other words, $f(\mathbf{p}(t) \parallel \mathbf{p}^O(t))$ computes the residual optimum pretending that each job exists in two separate copies, with remaining sizes $p_j(t)$ and $p_j^O(t)$, respectively. As mentioned before, this abstracts away the beautiful potential function used in [CGKM09] for unrelated machines.⁶

It is known [IMP11] that an online algorithm can be shown to be $O(1)$ -competitive (with some speed augmentation) if the potential satisfies the following:

- (P1). Φ does not increase when a job arrives or is completed by A or O .
- (P2). $\frac{d}{dt} \Phi(t) \leq -|A(t)| + O(1)|O(t)|$ at all the other times.

This is because $\Phi(t=0) = \Phi(t=\infty) = 0$ (either no jobs have arrived or all jobs have been completed) and due to (P1) and (P2), we know that $0 \leq \int_{t=0}^{\infty} \frac{d}{dt} \Phi(t) dt \leq -\int_{t=0}^{\infty} |A(t)| dt + O(1) \cdot \int_{t=0}^{\infty} |O(t)| dt$ (the integral is taken over the whole time except when jobs arrive or are completed.) Note that $\int_{t=0}^{\infty} |A(t)| dt$ is A 's objective since every job alive at time t incurs a unit cost at the time. Similarly, $\int_{t=0}^{\infty} |O(t)| dt$ is O 's objective. Thus we can establish A 's $O(1)$ -competitiveness.

⁴If \mathbf{z} is fractional, we can implement it by preemptive time sharing.

⁵Flow time is also known as response time or waiting time.

⁶The potential function in [CGKM09] looks somewhat different at first sight as it measures the aggregate increase of the residual optimum when each job in A is added to O , and vice versa. But, by rearranging terms carefully, one can observe that our potential function is essentially equivalent to theirs in the setting of unrelated machines.

Unlike [IMP11] that repeats trial-errors to find a potential function that satisfies these properties to analyze an online scheduling algorithm that a creative algorithm designer came up with, our approach *reduces the whole algorithm design and analysis to checking if $f(\mathbf{x})$ satisfies supermodularity*. More formally, let $g(Y) := f(\mathbf{x} \odot \mathbf{1}_Y)$ ⁷ be the residual optimum only considering jobs in Y when each job j 's (remaining) size is x_j . Supermodularity means $g(X \cup \{j\}) - g(X) \geq g(Y \cup \{j\}) - g(Y)$ for any j and $X \supseteq Y$. In other words, it implies that the increase in the residual optimum caused by the addition of a new job can only be larger if there are more jobs.

We show that we can obtain (P1) from the supermodularity of f by *crystallizing* the analysis of [CGKM09]. Further, (P2) follows from the nature of GD. In general, we can show that GD just follows the residual optimum schedule (until a new job arrives) and therefore decreases every alive job's projected completion time (counted from now) by one. Thus, the residual optimum (the minimum total completion time) decreases at a rate of $s|A(t)|$ when GD is given speed $s > 1$, i.e., $\frac{d}{dt}f(\mathbf{p}(t)) = -s|A(t)|$.⁸ By the nature of GD, this is also the maximum possible decrease rate, which is the speed times the number of jobs alive. By thinking of the processing being done on $\mathbf{p}(t) \parallel \mathbf{p}^O(t)$ as a feasible schedule with speed $s + 1$ (combining GD's speed s and the adversary's speed 1), we have $\frac{d}{dt}f(\mathbf{p}(t) \parallel \mathbf{p}^O(t)) \geq -(s + 1)(|A(t)| + |O(t)|)$. Thus, (P2) follows with $s = 1 + \epsilon$.

In summary, we demonstrated that GD can be naturally derived from the residual optimum and its competitiveness follows from the residual optimum's supermodularity. This argument extends to an arbitrary polytope scheduling problem (Theorem 3.2).

1.2.2 Substitutes Empower Gradient Descent (Theorems 3.3 and 3.4) Theorem 3.2 establishes supermodularity as a crucial structural property that ensures the effectiveness of GD. However, demonstrating supermodularity of the residual optimum is challenging for arbitrary problems, while trivial for the single-machine case. In fact, computing the residual optimum is NP-hard for many scheduling scenarios. Proving supermodularity without access to an efficient value oracle for the function appears practically impossible.

To address these challenges, we can first consider an approximate residual optimum, which can be efficiently obtained by solving a time-indexed LP. Time-indexed LPs are widely used in the literature, e.g. [LAHW97, AGK12]. We maintain our focus on single machine scheduling for illustration.

$$\min \sum_{j, \tilde{t} \geq 1} \frac{\tilde{t}}{p_j} z_{j\tilde{t}} \quad \text{s.t.} \quad \sum_j z_{j\tilde{t}} \leq 1 \quad \forall \tilde{t} \geq 1; \quad \sum_{\tilde{t} \geq 1} z_{j\tilde{t}} \geq p_j(t) \quad \forall j; \quad \mathbf{z} \geq 0$$

Here, we use \tilde{t} in the LP to distinguish the time variables⁹ from actual time t . Variable $z_{j\tilde{t}}$ implies that job j is processed at a rate of $z_{j\tilde{t}}$ at time \tilde{t} . The second constraint ensures that all jobs must be eventually completed. In general, the first constraint can be replaced with $\{z_{j\tilde{t}}\}_j \in \mathcal{P}$, which encodes each scheduling problem's constraints. The objective is fractional since when job j is processed at time \tilde{t} by one unit, we pretend $\frac{1}{p_j}$ fraction of it is completed by the algorithm. It is well known that an algorithm that is scalable for the fractional objective can be converted into one that is scalable for the integer objective (Lemma 2.1). Thus, it suffices to derive GD from the fractional residual optimum. We show that we can implement GD as follows: Solve the LP and follow the solution, i.e., process job j at a rate of $z_{j\tilde{t}}$ at time $t + \tilde{t}$, until a new job arrives. We show how to make the LP compact in Appendix C to make GD run in polynomial time.

However, we now face another challenge: The optimum LP solution lacks a nice structure. Despite this, how can we show the supermodularity of the optimum LP objective as a set function of jobs? To show supermodularity, we make a connection to market equilibria. To this end, by flipping the sign (and adding some large constants B), we can convert the problem into a maximization problem as follows. Now the goal becomes showing *submodularity* of the objective.

$$\max \sum_{\tilde{t} \geq 1} \sum_j \left(B - \frac{\tilde{t}}{p_j}\right) z_{j\tilde{t}} \quad \text{s.t.} \quad \sum_j z_{j\tilde{t}} \leq 1 \quad \forall \tilde{t} \geq 1; \quad \sum_{\tilde{t} \geq 1} z_{j\tilde{t}} \leq p_j(t) \quad \forall j; \quad \mathbf{z} \geq 0$$

⁷The operation \odot represents component-wise multiplication.

⁸Speed augmentation is a common beyond-worst-case model where the algorithm with slightly higher speed is compared to the adversary with speed 1.

⁹Here, a unit time is assumed to be sufficiently small.

We consider the following (Walrasian) market: Items correspond to the remaining job sizes. Each item j exists in $p_j(t)$ units. Each time \tilde{t} has an agent with a valuation function,

$$v_{\tilde{t}}(\mathbf{x}) := \sum_j \left(B - \frac{\tilde{t}}{p_j}\right) x_j,$$

where the function is defined over all $\mathbf{x} \geq 0$ such that $\|\mathbf{x}\|_1 \leq 1$.¹⁰ Suppose the items are priced at \mathbf{q} (each unit of job j is priced at q_j). To maximize her quasi-linear utility (valuation minus payment), the agent at time \tilde{t} buys one unit of the item/job with the maximum value of $B - \tilde{t}/p_j - q_j$.

A market equilibrium is an allocation of items to agents along with a price vector where every agent gets an assignment maximizing their utility and all items are sold out. The maximum total valuation of all agents is called the maximum social surplus, and it is well known that an *equilibrium results in the maximum social surplus* from LP duality.

With this interpretation, the question becomes whether the max social surplus is submodular as a set function of items/jobs. Gross substitute (GS) valuations come to the rescue here. As mentioned before, a valuation is GS if increasing one item's price does not decrease the demand for other items in the agent's optimal choice. It is trivial to see $v_{\tilde{t}}$ is GS: The agent buys the item $j = \arg \max_{j'} B - \tilde{t}/p_{j'} - q_{j'}$ and increasing any other job's price does not affect her choice; the GS condition trivially holds if we increase j 's price.

The beauty of GS lies in the fact that *if every agent has a GS valuation, then the market equilibrium exists* and can be found by a tatonnement process where prices are monotonically increased from zero ensuring every item is demanded by at least one agent. In such a market, the agents with GS valuations collectively behave as a single agent with GS valuation. This aligns with the fact that GS functions are closed under max-convolution [LLN06], resulting in *the max social surplus being GS*. Since *GS is a subclass of submodular functions* (but not vice versa) [GS99, LLN06], we can conclude that the max social surplus is submodular as a set function of items/jobs, as desired.

1.3 Applications We outline some applications of GD. We give scalable algorithms for all these applications—by using Theorem 3.3 for the first two and by using Theorem 3.4 for the third. The last application uses Theorems 3.2 and 3.3 with a small tweak. The details, including how the scheduling problems are captured by the polytope scheduling (or equivalently multidimensional scheduling) are deferred to Section 5. *No scalable algorithms were known for any applications below except unrelated machines, prior to our work. In particular, no positive algorithms existed for generalized flow scheduling and speed-up curves on partitioned resources.*

Matroid Scheduling [IKM18, IMPP19]. In this problem, we are given a matroid $\mathcal{M} = (J, \mathcal{I})$ where \mathcal{I} is a collection of job sets. The pair $\mathcal{M} = (J, \mathcal{I})$ is called a matroid if it satisfies: if $I' \subseteq I$ and $I \in \mathcal{I}$, then $I' \in \mathcal{I}$; and if $|I| < |I'|$ and $I, I' \in \mathcal{I}$, then there exist $j \in I' \setminus I$ such that $I \cup \{j\} \in \mathcal{I}$. For numerous examples of matroids, see Chapter 8 of [Sch03]. At any time instant t , a feasible schedule is an independent set I in \mathcal{I} . If we schedule an independent set I of jobs, then each job in I gets processed at a rate of 1. We give a $(1 + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive algorithm. Previously, [IKM18] gave a $(e + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive algorithm.

Matroid scheduling encompasses many scheduling problems. As discussed earlier, a uniform matroid of rank m can model parallel identical machines (at every time step, we can process up to m jobs). A gammoid captures single-source routing in a directed graph with a single source where jobs are processed when there are vertex-disjoint paths from the source to their corresponding nodes. A cographic matroid captures edge maintenance scheduling in an undirected graph. When performing service on an edge, it becomes unusable at the moment. However, concurrent maintenance work on some edges is possible if the remaining graph stays connected without those edges. In this scenario, maintenance jobs arrive at edges.

Generalized Flow Scheduling. This generalizes the gammoid scheduling, a special case of the matroid scheduling. We are given a network $G = (V, E)$, where $J \subseteq V$ is the set of source nodes. Arc e has capacity u_e and flow gain factor γ_e ; a flow of value f becomes $\gamma_e f$ after flowing through e . Gain factors distinguish generalized flow from standard flow and are extremely useful for modeling *heterogeneity*, as we can have a different factor for each edge. Job j gets processed at a rate equal to its (outgoing) net flow. It is critical for cloud computing to manage heterogeneous computing resources connected via a network; see [JS15]. A job gets processed by

¹⁰Technically we need to extend the domain of $v_{\tilde{t}}$ to encompass all non-negative vectors $\mathbf{x} \geq 0$. However, we will disregard this for the sake of illustration in this section.

communicating with the resources and the heterogeneous benefit between jobs and resources can be captured by edge gain factors, e.g. [IM15]. We give a $(1 + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive algorithm. No prior work exists for generalized flow scheduling.

Speed-up Curves on Partitioned Resources. This generalizes the arbitrary speed-up curves model on uniform processors [DGBL00, ECBD03, EP12], which elegantly models how the parallelization effect degrades as we use more processors to speed-up processing a job. There are D different divisible resources, where the i th resource exists in m_i units. At every time, resources are allocated to jobs. Each job has a valuation function of the following form: Resources are partitioned into k disjoint groups $\mathcal{G} = \{G_1, \dots, G_k\}$; partitioning can be different for each job. Each group $G_i \in \mathcal{G}$ is associated with a monotone univariate concave function $g_i : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ with $g_i(0) = 0$. Then, the processing rate of job j is $v_j(\mathbf{x}_j) = \sum_{i=1}^k g_i(\sum_{d \in G_i} a_{jd} x_{jd})$, where $a_{jd} \geq 0$, when j receives $\mathbf{x}_j = (x_{j1}, x_{j2}, \dots, x_{jD})$ resources. Concave functions are used to model the *diminishing returns of parallelization* as we add more computing resources. The former arbitrary speed-up curves is a special case of this model when $D = 1$. We give a $(1 + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive algorithm. No prior work exists for this problem.

To see why this general model is useful, suppose $D = 2$ and consider three types of valuation functions: x_{j1}^5 , x_{j2}^{75} , and $(x_{j1} + x_{j2})^5$, for an example. The first means the job can only use processors in G_1 , and the second means the job can only use those in G_2 , but the third can use any processors. This can model the *restriction of processors* available for each job's processing. CPU partitioning is common in practice [CPU].

Unrelated Machines [LST90, SS99, Sku01, SS02, CGKM09, AGK12, DH14]. There is a set of parallel machines. Each job j gets processed at a rate of λ_{ij} when scheduled on machine i ; this is equivalent to j having a processing time p_j/λ_{ij} if scheduled only on machine i . Each job can be processed by at most one machine at any point in time.

We first *reproduce* the seminal result in [AGK12], which gave a $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive algorithm for minimizing total weighted flow time (Section 5.7). We use the same algorithm of [AGK12], which immediately dispatches an arriving job to one of the machines and keeps the job on the same machine until its completion. Perhaps, the analysis of [CGKM09, AGK12] becomes more transparent through the lens of our framework, supermodularity and gradient descent.

While [AGK12] gave essentially the best possible result and we can recover it, the work left an intriguing question from a technical point of view. Although immediate-dispatch and non-migration are highly desirable, they were strongly required in the analysis; it was not known whether we could still achieve a competitive algorithm if we migrated jobs in the middle of the execution for better load balancing. This has been puzzling in the research of online scheduling algorithms.

Our approach answers this question positively by providing a scalable algorithm that changes job assignments when a new job arrives to achieve better load balancing. Furthermore, our algorithm is purely *Markovian*, meaning it only needs to remember the amount of jobs processed, eliminating the need to track job assignments. Specifically, we present a $(1 + \epsilon)$ -speed, $O(1/\epsilon^2)$ -competitive¹¹ algorithm for unweighted total flow time (Section 5.5) and a $(1 + \epsilon)$ -speed, $O(1/\epsilon^3)$ -competitive algorithm for weighted total flow time (Section 5.6). While these results have a worse competitive ratio, we believe the Markovian property has the potential for broader applications; see Section 1.5.

1.4 Comparison with the Previous Work As mentioned, Proportional Fairness (PF) [Nas50, KMT98, GZH⁺11, JV10] is the only online scheduling meta-algorithm known other than our gradient descent (GD). PF finds a fair allocation in Fisher markets, where each agent (job j) starts with an endowment (job j 's weight w_j) and buys some resources to maximize its valuation (how fast it gets processed at the moment). PF solves $\max \sum_j w_j \log v_j(\mathbf{y}_j)$ s.t. $\sum_j \mathbf{y}_j \leq 1$, $\mathbf{y} \geq 0$ where j has valuation $v(\mathbf{y}_j)$ when it receives \mathbf{y}_j resources from the supply vector of divisible resources.¹² The solution can be viewed as an equilibrium where the market is cleared under the corresponding dual prices of the resources. Note that PF finds an instantaneous fair allocation without factoring in job sizes, hence is non-clairvoyant. The main contribution of [IKM18] lies in showing that PF is $O(1_\epsilon)$ -competitive with $(e + \epsilon)$ -speed for the total weighted flow time objective for scheduling problems when the

¹¹For the restricted assignment case, i.e., $\lambda_{ij} \in \{0, 1\}$ for all i, j , we can get an $O(1/\epsilon)$ -competitive algorithm when jobs are unweighted (Section 5.4).

¹²Equivalently, the scheduling problem can be expressed as a polytope \mathcal{P} that describes how fast jobs can be processed, i.e., $\mathcal{P} := \{x_j = v_j(\mathbf{y}_j) \mid \sum_j \mathbf{y}_j \leq 1, \mathbf{y} \geq 0\}$.

allocation under PF is monotone—a job’s speed/share can only decrease when other jobs arrive.

Although PF is a remarkable meta-algorithm generalizing RR (Round-Robin), it exhibits limitations from an algorithm design point of view. It is because we lose all the nice properties of PF when we deviate from the equilibrium it finds. Thus, it is not clear how to reduce the speed requirement of the algorithm; even RR requires $(2 + \epsilon)$ -speed over the adversary for competitiveness. Moreover, it disregards job sizes in scheduling decisions, a crucial aspect for designing machine learning-augmented algorithms that leverage job size estimates, a topic of recent significant interest [PSK18, IKMQP21, ALT21, ALT22, LM22].

Driven by these considerations, we investigate GD as a meta-algorithm, as the spirit of GD is often ingrained in online scheduling. Furthermore, unlike PF, GD offers flexibility. Implementing GD requires an estimate of the remaining cost. One common method of obtaining this estimate is to first fix a candidate algorithm and derive a reasonably accurate estimate of its remaining cost, which is converted into a potential function of a nice mathematical form [IMP11]. However, this approach demands algorithmic creativity, which involves a lot of trial and error.

Alternatively, one can utilize the (approximately) optimal remaining cost, also known as the residual optimum. Note that the residual optimum is not tied to specific algorithms. However, computing the residual optimum is often NP-hard, rendering it challenging to understand its evolution. One approach is employing an LP relaxation, but this results in an LP solution, which commonly lacks structure.

Our work draws upon critical ideas from the seminal work of [CGKM09], which presented the first scalable algorithm for unrelated machines; see 1.3 for more details. They introduced an elegant potential function, and our meta-potential function is its generalization.

Generalizing their work poses challenges. Their algorithm executes SRPT (or its weighted version) on each machine following the initial assignment, and SRPT admits a closed-form expression for the residual optimum. Their potential function computes the residual optimum for a mixture of jobs from the algorithm and the adversary. However, determining how to proceed when the residual optimum lacks a convenient closed-form mathematical expression remains unclear.

We address the challenges as follows. First, we abstract away the potential function argument presented in [CGKM09] and demonstrate that their potential function can be generalized to work solely with supermodularity when gradient descent is employed. Furthermore, by establishing a novel connection to Walrasian markets, we simplify the task of verifying whether the residual optimum is supermodular.

The connection we make between gradient descent to Walrasian markets is less obvious than PF’s connection to the Fisher markets. It is well-known that PF finds a market-clearing equilibrium in the Fisher market, but it is unclear how markets are related to the residual optimum. In Walrasian markets, agents have no endowments and they try to maximize their quasi-linear utility, which is their valuation minus the payment for the resources bought. We use gross substitutes¹³ and linear substitutes to find sufficient conditions to obtain supermodularity.

Finally, PF and GD are somewhat complementary to each other. PF makes scheduling decisions without knowing job sizes but uses $(e + \epsilon)$ speed to be $O(1)$ -competitive. On the other hand, GD only needs $(1 + \epsilon)$ speed augmentation. Further, there are problems for which one is $O(1)$ -competitive (with $O(1)$ -speed augmentation), but not the other. For example, for broadcast scheduling PF is $O(1)$ -competitive, but not gradient descent (Appendix B). For unrelated machines, PF is believed not to be $O(1)$ -competitive [IKM18], but we show gradient descent is (Sections 5.5, 5.6, and 5.7).

1.5 Future Work In this paper, we show a reduction from designing online scheduling problems to the offline problem of finding an LP relaxation whose optimum satisfies supermodularity. In particular, we show if the underlying valuation function subject to the polytope characterizing the scheduling constraints at each time satisfies substitute properties, gradient descent is scalable. We discuss several open problems below.

- Gradient descent does not yield an $O(1)$ -competitive algorithm for all scheduling problems as mentioned above (Appendix B). Can we find other sufficient conditions for GD to be $O(1)$ -competitive? For example, can we design algorithms for the intersection of two matroids? The matroid intersection is known not to be a gross substitute [Mur22]. Here, it may increase the required speed or competitive ratio.
- Can we extend gradient descent to ℓ_k -norms of flow time? Although a naive extension of our framework

¹³It is important to note that they have different definitions in Fisher and Walrasian markets.

to ℓ_k -norms seems to create an undesirable increase of the potential due to job ages, we believe it should not be a major roadblock. No meta-algorithms have been considered for ℓ_k -norms of flow. We have some preliminary results for this objective.

- Is there an online algorithm for minimizing ℓ_k -norms of flow time where the competitive ratio has no dependency on k ? For unrelated machines, there exist $(1 + \epsilon)$ -speed $O(k)$ -competitive immediate-dispatch algorithms [IM11, AGK12]. However, it is known that any immediate dispatch algorithm must have a linear dependency on k [IM11, AGK12]. Since there exists a scalable algorithm for the maximum unweighted flow time with no dependency on k , it is plausible to remove the dependency by considering migratory algorithms [ABF⁺17]. As discussed, our algorithm allows migration, and thus has the potential to solve this problem.
- Can we develop another nonclairvoyant meta-algorithm besides PF? PF generalizes Round Robin, which requires at least $(2 + \epsilon)$ -speed to be $O(1)$ -competitive. However, Shortest Elapsed Time First (SETF)¹⁴ is known to be scalable in the single-machine setting and uses information about how much jobs have been processed. Can we generalize SETF in a similar way that SRPT and Round Robin were generalized to gradient descent and PF?

2 Preliminaries

2.1 Common Notation We use 1_X to denote a binary vector corresponding to X . We use bold fonts for vectors. For two vectors \mathbf{x}, \mathbf{y} , $\mathbf{x} \wedge \mathbf{y}$ takes the component-wise min of \mathbf{x} and \mathbf{y} . Similarly, $\mathbf{x} \vee \mathbf{y}$ takes the component-wise max of \mathbf{x} and \mathbf{y} . We let $\mathbf{x} \odot \mathbf{y}$ denote the component-wise multiplication of \mathbf{x} and \mathbf{y} . When v is a set function, we will interchangeably use $v(X)$ and $v(1_X)$ for a set X .

2.2 Problem Definition and Scheduling Primitives For easy reference, we reproduce the definition of the polytope scheduling below. All scheduling problems we will study fall into the polytope scheduling problem, or equivalently multidimensional scheduling.

Polytope Scheduling (Feasible Processing Rate View). Most preemptive scheduling problems with fixed computing resources can be represented as an instance of the polytope scheduling problem (PSP) [IKM18]. In the PSP, we are given a set of constraints that form a downward-closed¹⁵ polytope \mathcal{P} (more generally a convex body), which encodes the feasible processing rate vector at each time: each job j can be processed at a rate of z_j if and only if $\mathbf{z} := \{z_j\}_j \in \mathcal{P}$. For example, we have $\mathcal{P} = \{\mathbf{z} \mid \sum_j z_j \leq 1 \text{ and } \mathbf{z} \geq 0\}$ in the single machine scheduling, which says jobs can be processed at a rate of up to 1 in total. We assume that \mathcal{P} remains fixed over time.

Multi-dimensional Scheduling (Resource View). Equivalently, the PSP can be viewed as multidimensional scheduling where each dimension corresponds to a distinct resource. There exists a set of D divisible resources or a continuous supply vector 1 ,¹⁶ which are replenished at each time. Each job j is associated with a concave valuation function, $u_j : [0, 1]^D \rightarrow \mathbb{R}_+$, which denotes how fast it gets processed. A feasible allocation is constrained by $\sum_j \mathbf{y}_j \leq 1$: Job j gets assigned a resource vector \mathbf{y}_j and gets processed at a rate of $u_j(\mathbf{y}_j)$. We will call this resource view and the above feasible processing rate view.

We consider the clairvoyant scheduling setting: Each job j arrives at time r_j with (original) processing size p_j and weight w_j . Note that the online scheduler learns about job j , including w_j and p_j , only when it arrives. A job is completed when it has been processed by p_j units since its arrival time r_j . Preemption is allowed, meaning a job can be paused and resumed later. Our goal is to minimize the following objective function.

Total Weighted Flow Time. For integral flow time objective, an individual job j 's flow-time is defined as $F_j = C_j - r_j$ which is the difference between its arrival time r_j and completion time C_j ; we may add σ to C_j as superscript to make clear the schedule σ considered. Note that C_j is defined the min time t' such that $\int_{t=r_j}^{t'} z_j(t) dt = p_j$, where $z_j(t)$ is the processing rate of job j at time t in the schedule σ . Then, the total weighted integral flow time is $\sum_{j \in J} w_j F_j$, where w_j is the weight of job j and J is the entire set of jobs arriving to be completed.

¹⁴In the single machine scheduling, SETF processes the job that has been processed the least.

¹⁵If $\mathbf{z}' \leq \mathbf{z}$ and $\mathbf{z} \in \mathcal{P}$, then $\mathbf{z}' \in \mathcal{P}$.

¹⁶We can assume wlog that every resource exists by one unit by scaling.

Speed Augmentation. A large number of scheduling problems do not admit $O(1)$ -competitive algorithms. In such cases, speed augmentation is commonly used for beyond-worst-case analysis. Here, the online scheduling algorithm runs with extra speed compared to the optimal scheduler. If an algorithm can process $s > 1$ times more than the adversary can process at a time and has an objective at most c times than the adversary does, we say that the algorithm is s -speed c -competitive. Intuitively, this implies that the algorithm can handle $1/s$ -fraction of what the optimum solution can handle [KP00]. Thus, an algorithm that is $O(1_\epsilon)$ -competitive with $(1 + \epsilon)$ -speed for any $\epsilon > 0$ is of fundamental importance and termed *scalable*.

Total Fractional Weighted Flow Time. In many scenarios, we will consider the fractional flow time which is a relaxation of integral flow time. The fractional weighted flow time objective is defined to be $\sum_{j \in J} \int_{t \geq 0} \frac{w_j}{p_j} t z_j(t) dt$. In other words, we view $\{z_j(t)/p_j\}_t$ as a distribution when the job is completed. By definition, it is obvious that a job's fractional flow time is no greater than its integer flow time in any fixed schedule. It is easier to minimize fractional flow time than integer flow time. Fortunately, with an extra speed augmentation, we can convert an algorithm that is good for fractional flow time into one that is good for integer flow time.

Lemma 2.1 ([CGKM09, IMP11, IKM18]). *For any polytope scheduling problem, we can convert an online algorithm whose total fractional weighted flow time is C into an online algorithm with (integer) weighted flow time $O(C/\epsilon)$ given $(1 + \epsilon)$ -speed. Therefore, if there is an online scheduling algorithm that is c -competitive with $s > 1$ speed for total fractional weighted flow time, then for any $\epsilon > 0$, there exists an algorithm that is $s(1 + \epsilon)$ -speed $O(c/\epsilon)$ -competitive for total (integral) weighted flow time.*

For a fixed schedule σ , we will let $\sigma(t) := \{j \mid r_j \leq t < C_j^\sigma\}$ denote the set of jobs alive at time t in σ . We will often use A and O to denote our algorithm's schedule and the adversary's schedule, respectively. As an alive job j adds w_j at each time to the total weighted flow time objective, we immediately have the following.

Proposition 2.2. *We have $\int_{t=0}^\infty W^\sigma(t) dt = \sum_j w_j (C_j^\sigma - r_j)$, where $W^\sigma(t) := \sum_{j \in \sigma(t)} w_j$ is the total weight of jobs alive at time t in σ .*

This observation also extends to the fractional objective. Let $p_j^\sigma(t)$ be j 's remaining size at time t in schedule σ .

Proposition 2.3. *We have $\int_{t=0}^\infty \tilde{W}^\sigma(t) dt = \sum_j w_j \int_{t=r_j}^\infty \frac{z_j^\sigma(t)}{p_j} t dt$, the total fractional weighted flow time of σ , where $\tilde{W}^\sigma(t) := \sum_{j \in \sigma(t)} w_j \frac{p_j^\sigma(t)}{p_j}$ is the total fractional weight of jobs alive at time t in σ .*

2.3 Gross Substitutes for Indivisible Resources We first consider valuation functions for indivisible items. Note that we consider the Walrasian market model throughout this paper, which is different from the Fisher market, as discussed in Section 1.4. Formally, the Walrasian market is defined as follows.

Definition 2.4 (Walrasian Market). *A Walrasian market consists of a set of n indivisible items and m agents where each agent $i \in [m]$ has a valuation function $v_i : 2^{[n]} \rightarrow \mathbb{R}$. Each item $j \in [n]$ is priced at $q_j \in \mathbb{R}_+$. Then, the market gives a disjoint allocation $\{S_i\}_{i \in [m]}$ of items over agents. We define the social surplus of the market as $\sum_{i \in [m]} v_i(S_i)$.*

In the Walrasian market, each agent has a preference for bundles of items specified by the quasi-linear utility function, which can be formally defined by the demand correspondence as follows.

Definition 2.5 (Demand Correspondence). *Given a set function $v : 2^{[n]} \rightarrow \mathbb{R}$ and a price vector $\mathbf{q} \in \mathbb{R}_+^n$, we define the demand correspondence (or set) $\mathcal{D}(v, \mathbf{q})$ as the family of sets that maximize the quasi-linear utility of the agent:*

$$\mathcal{D}(v, \mathbf{q}) := \arg \max_X \{v(X) - \mathbf{q} \cdot \mathbf{1}_X\}.$$

In other words, $\mathcal{D}(v, \mathbf{q})$ represents the set of item subsets that maximize our total happiness (the difference between our valuation and the total cost) under the price vector \mathbf{q} , given our valuation function v . The Walrasian equilibrium asks that every agent maximizes its utility function and all items are allocated. Formally, the Walrasian equilibrium is defined as follows.

Definition 2.6 (Walrasian Equilibrium). *Given a Walrasian market with n indivisible items and m agents with valuations $\{v_i\}_{i \in [m]}$, a Walrasian equilibrium consists of a partition $\{S_i\}_{i \in [m]}$ of n and a price vector $\mathbf{q} \in \mathbb{R}_+^n$ such that $S_i \in \mathcal{D}(v_i, \mathbf{q})$ for all $i \in [m]$.*

It is known that an equilibrium exists if all agents' valuation function satisfies the GS property. GS is an important class of valuation functions [KC82].

Definition 2.7 (Gross Substitutes). *A valuation function $v : 2^{[n]} \rightarrow \mathbb{R}$ is gross-substitute (GS) if for any price vectors $\mathbf{q}' \geq \mathbf{q} \geq 0$ and any $X \in \mathcal{D}(v, \mathbf{q})$, there is a set $Y \in \mathcal{D}(v, \mathbf{q}')$ such that $X \cap \{j : q_j = q'_j\} \subseteq Y$.*

In other words, a GS valuation means that if we increase an item j 's price, we can update an optimum allocation without dropping any other items. Intuitively, this is possible because j can be substituted with other items unlike some items behaving like a bundle, e.g. milk and cereal.

It is known that GS is a strict subclass of submodular functions [GS99, LLN06].

Definition 2.8. *A valuation function $v : 2^{[n]} \rightarrow \mathbb{R}$ is submodular if for all $X, Y \subseteq [n]$, $v(X) + v(Y) \geq v(X \cap Y) + v(X \cup Y)$. Alternatively, it is submodular if $v(X \cup \{e\}) - v(X) \leq v(Y \cup \{e\}) - v(Y)$ for all $X \supseteq Y$ and $e \in [n]$. Symmetrically, v is supermodular if $-v$ is submodular.*

Lemma 2.9 ([GS99]). *A valuation function v that satisfies the GS property is submodular.*

Furthermore, Lehmann, Lehmann and Nisan [LLN06] showed GS valuations are closed under convolution, meaning that the aggregate max-valuation of GS valuations is GS.

Lemma 2.10. *The class of gross-substitute valuations is closed under convolution, i.e., for any two gross-substitute valuations v_1 and v_2 , $v_1 \oplus v_2$ is gross substitutes, where*

$$v_1 \oplus v_2(S) = \max_{T \subseteq S} \{v_1(T) + v_2(S \setminus T)\}$$

It is known that a Walrasian equilibrium exists when all agents have GS valuations, and a Walrasian equilibrium results in the max social surplus [KC82, GS99]. Thus, the lemma can also be interpreted as stating that the social surplus of a Walrasian market exhibits GS if all agents have GS valuations.

2.3.1 Examples We state a few examples of GS valuation functions. For a comprehensive list, see [ST15].

- Symmetric concave functions: $f : \{0, 1\}^n \rightarrow \mathbb{R} \cup \{\infty\}$ is a symmetric concave function if there exists a concave function $\varphi : \mathbb{R}_+ \rightarrow \mathbb{R}$ s.t. $f(\mathbf{x}) = \varphi(\sum_{i=1}^n x_i)$.
- Laminar concave functions: A set family $\mathcal{T} \subseteq 2^N$ is laminar if $X \cap Y = \emptyset$, $X \subseteq Y$ or $X \supseteq Y$ for every $X \neq Y \subseteq \mathcal{T}$. Define $\varphi_Y : \mathbb{R}_+ \rightarrow \mathbb{R}$ to be a univariate concave function. Then the function given by $f(x) = \sum_{Y \in \mathcal{T}} \varphi_Y(\sum_{i \in Y} x_i)$ is called a laminar concave function.
- Maximum weight bipartite matching: Let $G = (U, W; E)$ be a weighted bipartite graph with an edge cost function $w : E \rightarrow \mathbb{R}$. Define $f : \{0, 1\}^n \rightarrow \mathbb{R} \cup \{-\infty\}$ to be the maximum weight bipartite matching w.r.t a vertex set $X \subseteq U$ by

$$f(1_X) = \max \left\{ \sum_{e \in M} w(e) \mid M \subseteq E \text{ is a matching in } G \text{ covers all vertices in } X \right\}$$

- Weighted rank functions of matroids: Let $\mathcal{M} = (E, \mathcal{I})$ be a matroid (see Section 1.3 for definition), where E is a universe of n elements, and $\mathcal{I} \subseteq 2^E$ is a collection of independent sets. Let $w : E \rightarrow \mathbb{R}_+$ be a weight function. Then define $f : \{0, 1\}^n \rightarrow \mathbb{R}_+$ to be the weighted rank function w.r.t an element set $X \subseteq E$, i.e.,

$$f(1_X) = \max \left\{ \sum_{i \in Y} w(i) \mid Y \subseteq X, Y \in \mathcal{I} \right\}$$

2.4 Linear Substitutes for Divisible Resources Since we consider preemptive schedules where jobs can be processed by fractional amounts at a time and we will handle heterogeneous resources, we need a continuous extension of GS and Walrasian markets where resources to be allocated are divisible. Continuous definitions of gross substitutes, termed linear substitutes (LS), were introduced by Milgrom and Strulovici [MS09]. Interestingly, although the continuous setting does not inherit all properties that are guaranteed in the discrete setting [MS09, ST15], LS still offers some useful properties we can leverage. In particular, LS valuations also are closed under convolution (or equivalently called aggregation) and are a subclass of submodular functions.¹⁷

Definition 2.11 (Linear Substitutes). $v : \prod_{i \in [n]} [0, L_i] \rightarrow \mathbb{R}$ is a linear-substitute valuation if for any price vector \mathbf{q} and \mathbf{q}' , then whenever $q_j \leq q'_j$, $q_k = q'_k$ for all $k \neq j$ and $X \in \mathcal{D}(v, \mathbf{q})$, there is a $Y \in \mathcal{D}(v, \mathbf{q}')$ such that $Y_k \geq X_k$ for all $k \neq j$.

Lemma 2.12 ([MS09]). The class of linear-substitute valuations is closed under aggregation, i.e., for any linear-substitute valuations v_1, \dots, v_n , v is linear substitutes, where

$$v(\mathbf{x}) = \max \left\{ \sum_{i \in [n]} v_i(\mathbf{x}_i) : \sum_{i \in [n]} \mathbf{x}_i = \mathbf{x}, \mathbf{x}_i \geq 0 \ \forall i \in [n] \right\}$$

Similar to GS, this lemma implies that the social surplus is LS if all the agents have LS valuations in the Walrasian market considered.

To verify the LS property, we can check the submodularity of its dual profit function.

Lemma 2.13 ([MS09]). $v : \mathbb{R}^n \rightarrow \mathbb{R}$ is a linear-substitute valuation iff π is submodular, where

$$\pi(\mathbf{q}) := \max_{\mathbf{x} \in \mathbb{R}^n} v(\mathbf{x}) - \mathbf{q} \cdot \mathbf{x}$$

We can easily extend the proof of GS implying submodularity (Lemma 5 in [GS99]) to LS implying submodularity; for completeness, we give the proof in Appendix A.

Lemma 2.14. Let $v : \mathbb{R}^n \rightarrow \mathbb{R}$ be a concave linear-substitute valuation. Then, $v(\mathbf{x})$ is submodular.

2.4.1 Examples We discuss some examples of linear substitutes, which will be used as building blocks in applying our theorems. Compared to gross substitutes, relatively fewer examples are known in the literature. We show that the concave closure of GS and generalized flow satisfy LS, which could be of independent interest; the proofs are deferred to Appendix A. For more examples, see Section 5 of [Mur22].

Theorem 2.15 (Concave Closure of Gross Substitutes is LS). Given a gross-substitute valuation $v : \{0, 1\}^n \rightarrow \mathbb{R}$, let $v^+(\mathbf{x}) := \max \{ \sum_S v(S) \lambda_S : \sum_S \lambda_S \mathbf{1}_S = \mathbf{x}, \sum_S \lambda_S = 1, \lambda_S \geq 0 \}$ be the concave closure of v . Then, $v^+(\mathbf{x}) : [0, 1]^n \rightarrow \mathbb{R}_+$ is linear substitute.

To keep the flow of presentation, we defer the formal definition of generalized flow to Section 5.

Theorem 2.16 (Generalized Flow is LS). For any cost vector \mathbf{c} , $v(\mathbf{x}) := \max_{\mathbf{z} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{z}$ for the \mathcal{P} in Eqn. (5.14), which is defined for generalized flow, is LS.

3 Formal Description of Our Results

3.1 Gradient Descent on the Residual Objective We begin by formally defining the residual objective at each time t , namely the minimum projected cost of the remaining schedule. Let $A(t) := \{j \mid r_j \leq t < C_j\}$ be the set of jobs alive at time t in our schedule. Let $p_j(t)$ denote j 's remaining size at time t ; by definition, $p_j(t) = 0$ for all $t \geq C_j$. Since the objective we consider is a linear combination of jobs' flow time, we can observe that $w_j(C_j - t)$, is exactly how much j will contribute to the objective from time t until it is completed at time C_j as j gets processed by $p_j(t)$ units by the algorithm during $[t, C_j]$. So, if we shift time by t for easy indexing, we

¹⁷For a continuous function v , submodularity is defined analogously to Definition 2.8: $v(\mathbf{x}) + v(\mathbf{y}) \geq v(\mathbf{x} \vee \mathbf{y}) + v(\mathbf{x} \wedge \mathbf{y})$. Further, v is supermodular iff $-v$ is submodular.

have the following residual scheduling problem: Each alive job that has arrived has a (remaining) size $p_j(t)$ and has arrival time 0 (in the shifted time horizon), and weight w_j . Noticing that completion time is equivalent to flow time when jobs arrive at time 0, the residual optimum can be expressed as follows.

$$\begin{aligned} \text{Residual Optimization Problem } \mathcal{I}_x^P: \quad f(\mathbf{x}) := \min_{j \in A(t)} w_j \tilde{C}_j \quad s.t. \quad (3.2) \\ \int_{\tilde{t} \geq 0}^{\tilde{C}_j} z_j(\tilde{t}) d\tilde{t} = x_j \quad \forall j \in A(t) \\ \mathbf{z}(\tilde{t}) \in \mathcal{P} \quad \forall \tilde{t} \geq 0, \end{aligned}$$

where we set $x_j = p_j(t)$ for all $j \in A(t)$. Note that we added tilde to C_j and t to make it clear that we are considering the shifted time horizon. The first constraint ensures that every job j must be processed by x_j units, and the second guarantees feasible scheduling at every time.

Let $\mathcal{I}_x^P(\mathbf{z})$ denote the objective of \mathcal{I}_x^P for \mathbf{z} . In other words, $\mathcal{I}_x^P(\mathbf{z})$ is the extra cost we should pay if we follow \mathbf{z} from time t , i.e., process each job j at a rate of $z_j(\tilde{t})$ at time $t + \tilde{t}$, assuming no more jobs arrive. If \mathbf{z} is infeasible, then let $\mathcal{I}_x^P(\mathbf{z}) = \infty$. Let $f(\mathbf{x}) := \min_{\mathbf{z}} \mathcal{I}_x^P(\mathbf{z})$, which we will term algorithm A 's (integral) residual optimum at time t .

We are now ready to formally define our gradient descent algorithm: At each time t we try to decrease $f(\mathbf{x})$ the most, i.e.,

$$\begin{aligned} \text{Gradient Descent on the Residual Optimum } f(\mathbf{x}) : \quad \min_{\mathbf{z}} \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{p}(t)} \cdot \mathbf{z} \quad (3.3) \\ \mathbf{z} \in \mathcal{P} \end{aligned}$$

We will let $\text{GD}^P(\mathbf{x})$ denote the decrease of the integral residual optimum due to the gradient descent, i.e., the optimum to Eqn. (3.3). However, we will see that if $\mathbf{z}(\tilde{t})$ for $\tilde{t} \geq 0$ is an optimum solution that achieves $f(\mathbf{p}(t))$, to implement GD, we can simply process jobs according to $\mathbf{z}(0)$ at time t . In fact, we will show that we can exactly follow the residual optimum schedule until a new job arrives (Lemma 4.3). For brevity, we may omit \mathcal{P} from notation.

Notably, we take this time-indexed view since it explicitly describes the residual optimum solution at each time. Let us illustrate this with a single machine scheduling example we discussed in Section 1.2: After solving $\mathcal{I}_{\mathbf{p}(t)}^P$ we obtain: $z_{\pi(j)}(\tilde{t}) = 1$ for $\tilde{t} \in [0, p_{\pi(1)}(t))$ only for $j = 1$ (0 for all other jobs), $z_{\pi(j)}(\tilde{t}) = 1$ for $\tilde{t} \in [p_{\pi(1)}(t), p_{\pi(1)}(t) + p_{\pi(2)}(t))$ only for $j = 2$, and so forth. By solving Eqn.(3.3), we obtain \mathbf{z} such that $z_{\pi(1)} = 1$ and $z_{\pi(j)} = 0$ for all $j \neq 1$. Note that \mathbf{z} is identical to $\mathbf{z}(0)$. This is not a coincidence. Lemma 4.3 says that we can follow $\mathbf{z}(\tilde{t})$ to implement GD: process job $\pi(1)$ during time $[t, t + p_{\pi(1)}(t))$ and job $\pi(2)$ during $[t + p_{\pi(1)}(t), t + p_{\pi(1)}(t) + p_{\pi(2)}(t))$, and so forth. This is exactly what SRPT does.

In the following, we define the key desired property we use in our paper: for any fixed remaining job sizes \mathbf{x} , the residual optimum viewed as a set function is supermodular. There is an extended notion of submodularity/supermodularity in the continuous setting which is needed for linear substitutes (Section 2.4). However, this notion of supermodularity is sufficient for our purpose here. For brevity, we may simply say supermodularity without explicitly mentioning ‘discrete.’

Definition 3.1. We say a function $g : \mathbb{R}_{\geq 0}^{|J|} \rightarrow \mathbb{R}$ is (discrete-)supermodular if for any fixed \mathbf{x} , $\bar{g}_x(Z) := g(\mathbf{x} \odot \mathbf{1}_Z) : 2^J \rightarrow \mathbb{R}$ is supermodular, i.e., $\bar{g}(U) + \bar{g}(V) \leq \bar{g}(U \cap V) + \bar{g}(U \cup V)$ for all $U, V \subseteq J$, where $\mathbf{1}_Z$ is the binary characteristic vector of Z for any $Z \subseteq J$ and \odot denotes component-wise multiplication.

We identify supermodularity as a key sufficient condition for the gradient descent on the optimum integral residual objective to give a scalable online algorithm by the following theorem.

Theorem 3.2 (Gradient Descent Desiderata for Integral Objective). *For a polytope scheduling problem with polytope \mathcal{P} , suppose $f(\mathbf{x}) := \min_{\mathbf{z}} \mathcal{I}_x^P(\mathbf{z})$ is discrete-supermodular. Then, for any $\epsilon > 0$, GD is $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive for minimizing weighted integral flow time for PSP \mathcal{P} .*

3.2 Substitutes Empower Gradient Descent Unfortunately, a large number of scheduling problems are NP-hard to optimize for the integral completion objective. Further, it is non-trivial to check if the residual

optimum objective is supermodular. Thus, to make the gradient descent framework more readily applicable, we consider a fractional version of the residual objective and show it satisfies supermodularity for a large class of scheduling problems related to substitute valuation functions. Recall that we can use a standard conversion to translate an algorithm that is scalable for fractional weighted flow into one that is scalable for integer weighted flow; see Lemma 2.1.

As discussed in Section 2.2, we can take the feasible processing rate view (polytope scheduling) or equivalently the resource view (multi-dimensional scheduling). In the following, we consider both views.

3.2.1 Feasible Processing Rate View The following is a well-known time-indexed LP [HSSW97], which we will use to measure our fractional residual objective. As before, we use \tilde{t} to distinguish the time used in the residual schedule from the global time t .

$$\begin{aligned} \text{Residual (Time-Indexed) LP } \mathcal{L}_{\mathbf{x}}^{\mathcal{P}}: \quad f(\mathbf{x}) := \min \sum_{j \in A(t)} \frac{w_j}{p_j} \int_{\tilde{t} \geq 0} \tilde{t} \cdot z_j(\tilde{t}) d\tilde{t} \quad (3.4) \\ \int_{\tilde{t} \geq 0} z_j(\tilde{t}) d\tilde{t} = x_j \quad \forall j \in A(t) \\ \mathbf{z}(\tilde{t}) \in \mathcal{P} \quad \forall \tilde{t} \geq 0 \end{aligned}$$

This LP tries to minimize total weighted fractional completion time. Gradient descent on $f(\mathbf{x}) := \min_{\mathbf{z}} \mathcal{L}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z})$ solves $\min \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{p}(t)} \cdot \mathbf{z}$ subject to $\mathbf{z} \in \mathcal{P}$ and processes jobs at time $t + \tilde{t}$ according to $\mathbf{z}(\tilde{t})$, but as before, we will show that we can follow the LP solution until a new job arrives.

Theorem 3.3 (Gradient Descent Works for Linear Substitutes in Feasible Processing Rate View). *For a polytope scheduling problem with polytope \mathcal{P} , suppose that for any $\mathbf{c} \in \mathbb{R}_+^{|J|}$, $v(\mathbf{x}) = \max_{0 \leq \mathbf{z} \leq \mathbf{x}, \mathbf{z} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{z}$ is linear substitute. Then, for any $\epsilon > 0$, the GD gives a $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive online algorithm for minimizing total fractional weighted flow time; thus, there exists a $(1 + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive algorithm for total weighted flow time.*

The theorem can be proved separately for gross substitutes (GS) for indivisible resources. However, we only state the theorem for linear substitutes (LS) because we can show that the concave closure of GS is LS (Theorem 2.15). The conceptual contribution of the theorem lies in connecting GD to substitute valuation.

The time-indexed LP could be exponentially large. In Appendix C, we show that we can consider an approximate residual optimum to obtain a polynomial time algorithm—assuming that we have an efficient oracle to test whether $\mathbf{z}' \in \mathcal{P}$ or not.

3.2.2 Resource View We now consider the other equivalent view where each job gets some resources at each time from a D -dimensional supply resource vector $\mathbf{1}$, which is replenished at every time. Recall that each job j is associated with a valuation function u_j . Let $\mathbf{y}_{j\tilde{t}}$ denote the resource vector job j receives at time \tilde{t} . Job j then gets processed at a rate of $u_j(\mathbf{y}_{j\tilde{t}})$. We will keep reserving \tilde{t} to refer to times in the residual schedule. The LP we solve is essentially identical to $\mathcal{L}_{\mathbf{x}}^{\mathcal{P}}$, but has more explicit constraints that resources cannot be over-allocated. Concretely, we use the following linear programming.

$$\begin{aligned} \text{(Resource View) Residual LP (or CP) } \mathcal{R}_{\mathbf{x}}: \quad f(\mathbf{x}) := \min \sum_j \frac{w_j}{p_j} \int_{\tilde{t} \geq 0} \tilde{t} \cdot u_j(\mathbf{y}_{j\tilde{t}}) d\tilde{t} \quad (3.5) \\ \int_{\tilde{t} \geq 0} u_j(\mathbf{y}_{j\tilde{t}}) d\tilde{t} = x_j \quad \forall j \\ \sum_j \mathbf{y}_{j\tilde{t}} \leq \mathbf{1} \quad \forall \tilde{t} \geq 0 \end{aligned}$$

We prove the following theorem.

Theorem 3.4 (Gradient Descent Works for Linear Substitutes in Resource View). *For a multi-dimensional scheduling problem with a supply vector $\mathbf{1}$ of divisible resources, suppose that for every job j , $u_j : [0, 1]^D \rightarrow \mathbb{R}_+$ is*

strictly monotone and strictly concave¹⁸ linear substitute and further satisfies the following:

1. For any $\mathbf{y} \in \mathcal{D}(u_j, \mathbf{b})$ and $\lambda \geq 1$, there exists $\mathbf{y}' \in \mathcal{D}(u_j, \lambda \mathbf{b})$ such that $\mathbf{y} \geq \mathbf{y}'$.
2. For any $\mathbf{y} \in \mathcal{D}(u_j, \mathbf{b})$ and $\mathbf{b}' \leq \mathbf{b}$, there exists $\mathbf{y}' \in \mathcal{D}(u_j, \mathbf{b}')$ such that $u_j(\mathbf{y}') \geq u_j(\mathbf{y})$.

Then, for any $\epsilon > 0$, the GD applied to $\mathcal{R}_{\mathbf{p}(t)}$ yields a $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive online algorithm for minimizing total fractional weighted flow time; thus, there exists a $(1 + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive algorithm for integer weighed flow time.

In the theorem, in addition to every job j having an LS valuation function, we require two additional properties. The first says that if we increase all prices by the same factor, then each job demands only less resources. The second says that if we increase resource prices, each job's optimal demand maximizing its quasi-linear utility should have a lower valuation. While these two properties are intuitive, not all LS valuations satisfy them¹⁹.

To prove the theorem, we show that the valuation functions considered here induce an LS function in the feasible processing rate view, which allows us to use Theorem 3.3. The high-level proof idea is as follows. Consider $v(\mathbf{x}) = \max_{0 \leq \mathbf{z} \leq \mathbf{x}, \mathbf{z} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{z}$ for a fixed \mathbf{c} . We want to show the valuation v is LS. Let \mathbf{q} be the job prices per unit. By processing job j by one unit, we obtain value c_j , yet have to pay the price q_j . Finding the demand set for v can be viewed as finding the maximum social surplus when each job (agent) j has valuation $(c_j - q_j)u_j$ in the resource view; let \mathbf{b} be the equilibrium prices of the D resources. Increasing q_j lowers $(c_j - q_j)u_j$, which has the same effect of increasing the resource prices \mathbf{b} for j by a certain factor $\lambda > 1$. Thus it results in resources being under-utilized due to the first property. Then, it is known that the equilibrium prices drop [AC04, MS09]. All jobs' valuation functions $(c_i - q_i)u_i$, except for j , remain the same. Therefore, they can only get processed more due to the second property.

4 Proofs of Main Theorems

4.1 Proof of Theorem 3.2 It is highly important to note that in the proof when a job j arrives, we pretend that GD and the optimum schedule each are given a *distinct* copy of j . Therefore, if $A(t)$ and $O(t)$ denote the sets of jobs that have arrived yet have not been completed at time t by GD and by the optimum schedule respectively, then $A(t) \cap O(t) = \emptyset$ always. We will use A and O to denote GD's schedule and the optimum schedule, respectively.

We use $\mathbf{x} \parallel \mathbf{y}$ to denote the concatenation of two vectors \mathbf{x} and \mathbf{y} . Let $p_j^A(t)$ and $p_j^O(t)$ denote the remaining size of job j under our algorithm GD and a fixed optimum schedule respectively. For brevity we let $\mathbf{p}^A(t) := \{p_j^A(t)\}_{j \in A(t)}$ and $\mathbf{p}^O(t) := \{p_j^O(t)\}_{j \in O(t)}$. For analysis, we will use the following potential function.

$$\Phi(t) = \frac{2}{\epsilon} \left(f(\mathbf{p}^A(t)) - \frac{1}{2} f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t)) \right)$$

Following the standard potential function argument in online scheduling [IMP11] we will show the following:

1. **Boundary Conditions:** $\Phi(0) = \Phi(\infty) = 0$. In other words, at the beginning when no jobs have arrived, and at the end when all jobs have been completed, the potential is 0. This is obvious because the residual optimum is 0 when there are no jobs to be processed and it is only concerned with processing jobs that are currently alive. Further, it's worth noting that jobs of zero remaining sizes have no effect on the residual optimum.
2. **Discrete Changes:** Φ does not increase when a job arrives or is completed by GD or the optimum schedule. To this end, we will crucially use supermodularity.
3. **Continuous Changes:** $\frac{d\Phi(t)}{dt} \leq -W^A(t) + O(\frac{1}{\epsilon})W^O(t)$, when A is given $1 + \epsilon$ speed and O 1-speed, where $W^A(t) := \sum_{j \in A(t)} w_j$ and $W^O(t) := \sum_{j \in O(t)} w_j$ denote the total weight of jobs alive at time t in GD's schedule and the optimum's, respectively.

¹⁸Under this assumption, the demand set is a singleton set.

¹⁹For example, $u(\mathbf{y}) := \max \sum_i \lambda_i y_i$ s.t. $\|\mathbf{y}'\|_1 \leq 1$ and $\|\mathbf{y}'\|_1 \leq \|\mathbf{y}\|_1$ is LS, but does not satisfy the first property.

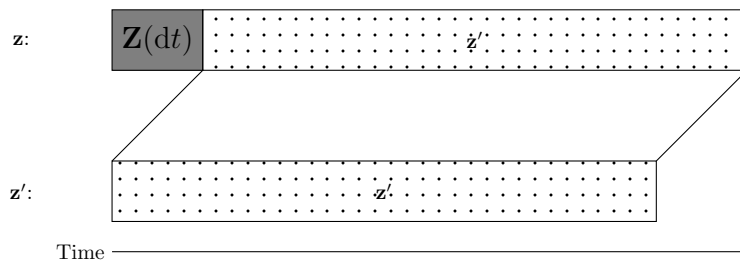


Figure 1: An illustration of the proof of Lemma 4.2 that shows the first direction ($\text{GD}(\mathbf{x}) \leq -\sum_j w_j$). The first row represents the residual schedule before GD's processing. The second row represents the schedule we consider after GD's processing for dt time units. GD processes the jobs $\mathbf{z}(\tau)$, $\tau \in [0, dt)$, which is colored dark grey. \mathbf{z}' is a schedule of the remaining sizes after processing sizes $\mathbf{Z}(dt)$. Since \mathbf{z}' is a feasible residual solution for remaining sizes $\mathbf{x}' = \mathbf{x} - \mathbf{Z}(dt)$, we can upper bound $\text{GD}(\mathbf{x})$ by comparing \mathbf{z} and \mathbf{z}' . Because \mathbf{z}' is a suffix of \mathbf{z} , shifted by dt time units, each job's completion time differs by exactly dt units in both schedules. This time difference translates directly to the difference in the objective function, which is the total weight of (alive) jobs.

Recall that $\text{GD}^{\mathcal{P}}(\mathbf{x}) := -\min_{\mathbf{z} \in \mathcal{P}} \nabla f(\mathbf{x}) \cdot \mathbf{z}$ denotes the change rate of the residual optimum for remaining sizes \mathbf{x} due to GD's processing. As mentioned earlier, the \mathcal{P} can be dropped for convenience, so $\text{GD}(\mathbf{x})$ denotes the same concept. If GD uses speed s , we denote it as $\text{GD}_s(\mathbf{x})$. Note that GD with s speed can process jobs at rates $s\mathbf{z}$ if $\mathbf{z} \in \mathcal{P}$.

Claim 4.1. *For any $s > 0$, we have $\text{GD}_s(\mathbf{x}) = -s\text{GD}(\mathbf{x})$.*

The following lemma proves that the change in the optimal residual objective value due to GD is precisely equal to the total weight of the uncompleted jobs. The intuition is simple: the residual optimal schedule remains optimal for minimizing total weighted completion time until a new job arrives. Consequently, every job's completion time in the residual schedule decreases by one in each time step. This establishes the " \leq " direction. The other direction can be proven similarly.

Lemma 4.2. *When the residual optimum is $f(\mathbf{x}) := \min_{\mathbf{z}} \mathcal{I}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z})$, we have $\text{GD}(\mathbf{x}) = -\sum_{x_j > 0} w_j$.*

Proof. Let $\mathbf{z}(\tau)$, $\tau \in [0, +\infty)$ be the optimal residual schedule for remaining size \mathbf{x} , i.e., $\mathbf{z} = \arg \min_{\mathbf{z}'} \mathcal{I}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z}')$. Consider the decrease of the residual optimum in dt units of time when we run GD. Let dt be an infinitesimally small time step such that no job is completed during time $[0, dt)$. Recall that GD schedules $\mathbf{z}(\tau)$, $\tau \in [0, dt)$. Let $\mathbf{Z}(\tau') := \int_{\tau''=0}^{\tau'} \mathbf{z}(\tau'') d\tau''$. Let $\mathbf{x}' = \mathbf{x} - \mathbf{Z}(dt)$ be the remaining sizes after processing \mathbf{z} in dt time. Note that $\mathbf{z}'(\tau) := \mathbf{z}(\tau + dt)$, $\tau \in [0, \infty)$ is a feasible schedule, for remaining job sizes \mathbf{x}' . Then we have $f(\mathbf{x}') \leq \mathcal{I}_{\mathbf{x}'}^{\mathcal{P}}(\mathbf{z}')$ because $f(\mathbf{x}')$ minimizes $\mathcal{I}_{\mathbf{x}'}^{\mathcal{P}}$ w.r.t. remaining sizes \mathbf{x}' , and \mathbf{z}' is a feasible schedule w.r.t. the same vector \mathbf{x}' . Thus we have

$$f(\mathbf{x}) - f(\mathbf{x}') \geq \mathcal{I}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z}) - \mathcal{I}_{\mathbf{x}'}^{\mathcal{P}}(\mathbf{z}') = dt \sum_{x_j > 0} w_j \quad (4.6)$$

This is because every job j s.t. $x_j > 0$ has a dt units larger completion time in \mathbf{z} than in schedule \mathbf{z}' and is still alive in the infinitesimally time step. Formally, j completes in \mathbf{z} at the maximum (supremum) time c such that $z_{cj} > 0$, and therefore it completes at time $c - dt$ in \mathbf{z}' since $\mathbf{z}(c) = \mathbf{z}'(c - dt)$.

To show the other direction, suppose GD schedules $\mathbf{z}''(\tau)$ for $\tau \in [0, dt)$ and let \mathbf{x}' be the resulting remaining job sizes. Let \mathbf{z}' be the optimal residual schedule for \mathbf{x}' ; thus we have $f(\mathbf{x}') = \mathcal{I}_{\mathbf{x}'}^{\mathcal{P}}(\mathbf{z}')$. Let \mathbf{z} be the schedule that first has \mathbf{z}'' for dt units of time and then exactly follows \mathbf{z}' . Since \mathbf{z} is a valid residual schedule for remaining sizes \mathbf{x} , we have $f(\mathbf{x}) \leq \mathcal{I}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z})$. Therefore,

$$f(\mathbf{x}') - f(\mathbf{x}) \geq \mathcal{I}_{\mathbf{x}'}^{\mathcal{P}}(\mathbf{z}') - \mathcal{I}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z}) = -dt \sum_{x_j > 0} w_j \quad (4.7)$$

As above, the equality holds as each job j 's completion time (such that $x_j > 0$) differs exactly by dt in the two schedules \mathbf{z} and \mathbf{z}' . This implies GD decreases the residual objective by at most $dt \sum_{x_j > 0} w_j$. Thus we have $\text{GD}(\mathbf{x}) = -\sum_{x_j > 0} w_j$, as desired. \square

The next lemma demonstrates that we can implement GD more efficiently by following the optimal residual schedule until a new job arrives. Upon a new job arrival, we recompute a new optimal residual schedule. The lemma is unnecessary if we are willing to compute the optimal residual schedule at every time step.

Lemma 4.3. *Let \mathbf{z} denote the optimum residual schedule for remaining sizes $\mathbf{p}(t)$. Suppose no jobs arrive during the time interval $[t, t']$. Then, we can assume wlog that GD processes jobs at rates $\mathbf{z}(\tilde{t})$ at each time $t + \tilde{t}$ for all $\tilde{t} \in [0, t' - t]$.*

Proof. The proof of this lemma is implicit in that of Lemma 4.2. Eqn. (4.6) shows that processing along \mathbf{z} , until a job is completed, is as effective as GD, which by definition is the most effective in decreasing the residual objective. Note that the remaining schedule must be an optimum residual schedule for the rest of the jobs. Then, we can recurse on the remaining sizes, until time t' when a new job arrives and we recompute the optimum residual schedule. \square

The following lemma bounds the potential change when a job arrives or is completed. This is where we use supermodularity.

Lemma 4.4. *If $\Phi(t)$ changes discontinuously due to a job's arrival or completion, the change is non-positive.*

Proof. If there is no arrival and completion, thanks to Lemma 4.3, it is straightforward to see $\Phi(t)$ is continuous in t and differentiable.

First, we observe that job completion in A 's schedule or the optimum solution's schedule has no effect on f and therefore on $\Phi(t)$. This is because once a job j 's remaining size becomes 0, then the residual optimum schedule does not process it. Thus, we only need to consider job arrival.

Suppose a new job j arrives at a fixed time t . For brevity, we drop t in the following. Let $\mathbf{1}_{j^A}$ denote the unit vector that has 1 only the position corresponding to j in A 's schedule. Define $\mathbf{1}_{j^O}$ analogously for the other copy of j in O 's schedule. Here j^A and j^O refer to j 's copy in the schedule of A and O , resp. Dropping t from the notation for brevity, $f(\mathbf{p}^A \parallel \mathbf{p}^O)$ increases by:

$$f(\mathbf{p}^A \parallel p_j \mathbf{1}_{j^A} \parallel \mathbf{p}^O \parallel p_j \mathbf{1}_{j^O}) - f(\mathbf{p}^A \parallel \mathbf{p}^O) = \bar{g}_{\mathbf{y}}(A \cup j^A \cup O \cup j^O) - \bar{g}_{\mathbf{y}}(A \cup O)$$

where $\mathbf{y} = \mathbf{p}^A \parallel p_j \mathbf{1}_{j^A} \parallel \mathbf{p}^O \parallel p_j \mathbf{1}_{j^O}$. Recall that the function $\bar{g}_{\mathbf{y}}(S)$ computes f assuming that each $j' \in S$ has size $\mathbf{y} \cdot \mathbf{1}_{j'}$ and every job not in S has zero size. Notably, we view f as a set function \bar{g} by fixing each job's (remaining) size. In the remaining proof, we use g instead of $\bar{g}_{\mathbf{y}}$ for notational brevity.

By supermodularity of g (due to the theorem condition), we have,

$$\begin{aligned} & g(A \cup j^A \cup O \cup j^O) - g(A \cup O) \\ & \geq g(A \cup j^A \cup j^O) - g(A) \\ & = g(A \cup j^A \cup j^O) - g(A \cup j^A) + g(A \cup j^A) - g(A) \\ & \geq g(A \cup j^O) - g(A) + g(A \cup j^A) - g(A) \\ & = 2(g(A \cup j^A) - g(A)), \end{aligned}$$

where the last inequality follows from the fact that j^O and j^A both have the same size p_j when j arrives.

Now, we shift to bounding the change of $f(\mathbf{p}^A)$, which increases by exactly

$$f(\mathbf{p}^A \parallel p_j \mathbf{1}_{j^A}) - f(\mathbf{p}^A) = g(A \cup j^A) - g(A)$$

Therefore, $\Phi(t)$'s change is at most $\frac{2}{\epsilon} (g(A \cup j^A) - g(A) - \frac{1}{2} \cdot 2(g(A \cup j^A) - g(A))) = 0$, as desired. \square

Lemma 4.5. *Consider any time t when no job arrival or completion occurs. If A and O are given s -speed and 1-speed respectively, $\frac{d}{dt} f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t)) \geq -(s+1)(W^A(t) + W^O(t))$.*

Proof. We observe that processing jobs in A and O can be thought as a feasible schedule with speed $s+1$ (A and O are given speeds s and 1 respectively). How much can one decrease $f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t))$ when given $s+1$ -speed? Lemma 4.2, together with Claim 4.1, implies that even GD, which is the most effective in decreasing the residual objective, can decrease at a rate up to the total weight of all jobs (of non-zero remaining sizes) times $s+1$ when given $s+1$ speed. Thus, the lemma follows. \square

Corollary 4.6. *Consider any time t when no job arrival or completion occurs. Then, $\frac{d}{dt}\Phi(t) \leq -W^A(t) + (1 + \frac{2}{\epsilon})W^O(t)$ when GD is given $1 + \epsilon$ -speed.*

Proof. From Lemmas 4.2 and 4.5, we have $\frac{d}{dt}\Phi(t) = \frac{d}{dt} \frac{2}{\epsilon} (f(\mathbf{p}^A(t)) - \frac{1}{2}f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t))) \leq \frac{2}{\epsilon} (-sW^A(t) + \frac{s+1}{2}(W^A(t) + W^O(t))) = -W^A(t) + (1 + \frac{2}{\epsilon})W^O(t)$, where $s = 1 + \epsilon$. \square

By Lemma 4.4 and the fact that $\Phi(T) = \Phi(0) = 0$, we have $\int_{t=0}^{\infty} \frac{d}{dt}\Phi(t)dt \geq 0$. By Corollary 4.6, we know that $\int_{t=0}^{\infty} W^A(t)dt \leq \frac{2+\epsilon}{\epsilon} \int_{t=0}^{\infty} W^O(t)dt$. Since the LHS and RHS are the total integral weighted flow time of GD and the optimum schedule respectively (Proposition 2.2), we have Theorem 3.2.

4.2 Proof of Theorem 3.3 We continue to use the same notation $\mathbf{p}^A(t)$ and $\mathbf{p}^O(t)$ as we used in the previous section. We use the same meta-potential function. But, here we use fractional completion time for the residual optimum, so we have $f(\mathbf{x}) := \min_{\mathbf{z}} \mathcal{L}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z})$ (see Eqn. (3.4)).

$$\Phi(t) = \frac{2}{\epsilon} \left(f(\mathbf{p}^A(t)) - \frac{1}{2}f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t)) \right)$$

As before, each job j has two distinct copies, which each appear in \mathbf{p}^A and \mathbf{p}^O .

We first establish f 's supermodularity.

Lemma 4.7. $f(\mathbf{x}) := \min_{\mathbf{z}} \mathcal{L}_{\mathbf{x}}^{\mathcal{P}}(\mathbf{z})$ is (discrete-)supermodular.

Proof. Showing the lemma is equivalent to showing $-f(\mathbf{x})$ is submodular. Then, the objective in Eqn. (3.4) becomes $\max \sum_{j \in A(t)} \int_{\tilde{t} \geq 0} -\frac{w_j \tilde{t}}{p_j} \cdot z_j(\tilde{t}) d\tilde{t}$. This is equivalent to $\max \sum_{j \in A(t)} \int_{\tilde{t} \geq 0} (B - \frac{w_j \tilde{t}}{p_j}) z_j(\tilde{t}) d\tilde{t}$ for any constant $B > 0$ due to the constraint $\int_{\tilde{t} \geq 0} z_j(\tilde{t}) d\tilde{t} = x_j$ for all $j \in A(t)$; also adding a constant does not change a function's submodularity. Further, if we let B be a sufficiently large constant, we obtain the following equivalent formulation.

$$\begin{aligned} & \max \sum_{j \in A(t)} \int_{\tilde{t} \geq 0} \left(B - \frac{w_j \tilde{t}}{p_j} \right) z_j(\tilde{t}) d\tilde{t} \\ & \int_{\tilde{t} \geq 0} z_j(\tilde{t}) d\tilde{t} \leq x_j \quad \forall j \in A(t) \\ & \mathbf{z}(\tilde{t}) \in \mathcal{P} \quad \forall \tilde{t} \geq 0 \end{aligned} \tag{4.8}$$

Note that although the first constraint does not require equality, due to the big constant B , the optimum solution to the above LP must process j to the full x_j units.

We now create a valuation function $v_{\tilde{t}}$ for each time \tilde{t} .

$$v_{\tilde{t}}(\mathbf{z}(\tilde{t})) := \max_{0 \leq \mathbf{y} \leq \mathbf{z}(\tilde{t}), \mathbf{y} \in \mathcal{P}} \sum_{j \in A(t)} \left(B - \frac{w_j \tilde{t}}{p_j} \right) y_j$$

In other words, here we view each time \tilde{t} as an agent²⁰ with valuation function $v_{\tilde{t}}$ and view each job j 's size/processing as a divisible resource, which exists by x_j units. When agent \tilde{t} receives processing units $\mathbf{z}(\tilde{t})$, it can process each job j and it would like to maximize an objective linear in \mathbf{y} subject to $\mathbf{y} \in \mathcal{P}$ and $\mathbf{y} \leq \mathbf{z}(\tilde{t})$.

By the precondition of the theorem, $v_{\tilde{t}}$ is LS. Then max LP objective is equivalent to

$$\max \int v_{\tilde{t}}(\mathbf{z}(\tilde{t})) d\tilde{t} \text{ s.t. } \int_{\tilde{t} \geq 0} \mathbf{z}(\tilde{t}) d\tilde{t} \leq \mathbf{x}$$

²⁰Technically, achieving this result requires discretizing the time horizon into a finite set of sufficiently small unit times and then taking the limit. However, to maintain clarity in the presentation, we omit this technical detail as it is straightforward.

Since this is a convolution of LS valuations, thanks to Lemma 2.12, we have the max LP optimum is LS w.r.t. \mathbf{x} . From Lemma 2.14, we know that it is submodular. Thus, we have shown f is supermodular. It is an easy exercise to show that this means f 's discrete-supermodularity (see Definition 3.1) using the fact that $1_{A \cup B} \odot \mathbf{x} = (1_A \odot \mathbf{x}) \vee (1_B \odot \mathbf{x})$ and $1_{A \cap B} \odot \mathbf{x} = (1_A \odot \mathbf{x}) \wedge (1_B \odot \mathbf{x})$. \square

The remaining analysis is almost identical to the proof of Theorem 3.3. We start the analysis by considering $\Phi(t)$'s discontinuous changes. The proof of the following lemma is identical to that of Lemma 4.4, which crucially used f 's supermodularity.

Lemma 4.8. *If $\Phi(t)$ changes discontinuously, the change is non-positive.*

Let $\tilde{W}^A(t) := \sum_{j \in A(t)} \frac{w_j}{p_j} p_j^A(t)$ denote the total fractional weight of jobs at time t in A 's schedule. Let $\tilde{W}^O(t)$ denote the corresponding quantity in O 's schedule. We will assume that GD follows the optimum residual schedule and show that no algorithm can decrease the residual optimum more effectively than GD.²¹ The following lemma quantifies the decrease rate of the (fractional) residual optimum.

Lemma 4.9. *For any time t when no job arrival or completion occurs, we have $\frac{d}{dt} f(\mathbf{p}^A(t)) = -s\tilde{W}^A(t)$ when we run GD with s speed. Furthermore, for any algorithm with s speed, we have $\frac{d}{dt} f(\mathbf{p}^A(t)) \geq -s\tilde{W}^A(t)$.*

Proof. Fix a time t and consider how much $f(\mathbf{p}^A(t))$ changes during $[t, t + dt)$ when we run GD with s -speed. Let $\mathbf{z} = \arg \min_{\mathbf{z}'} \mathcal{L}_{\mathbf{p}^A(t)}(\mathbf{z}')$. Note that j contributes to the objective $f(\mathbf{p}^A(t))$ by $\int_{\tau \geq 0} \frac{w_j}{p_j} \tau z_j(\tau) d\tau$. GD schedules $\mathbf{z}(\tau)$, $\tau \in [0, sdt)$ during time interval $[t, t + dt)$ as it is given s speed. Note that $\mathbf{z}(\tau)$, $\tau \in [sdt, \infty)$ is a feasible schedule (with 1-speed) for remaining job sizes $\mathbf{p}^A(t) - \mathbf{Z}(sdt)$ where $\mathbf{Z}(\tau') := \int_{\tau'' \geq 0} \mathbf{z}(\tau'') d\tau''$. Therefore, \mathbf{z}' , where $\mathbf{z}'(\tau) = \mathbf{z}(sdt + \tau)$, is a feasible solution to $\mathcal{L}_{\mathbf{p}^A(t) - \mathbf{Z}(sdt)}$.

Thus, $f(\mathbf{p}^A(t))$ decreases by

$$\begin{aligned} & f(\mathbf{p}^A(t)) - f(\mathbf{p}^A(t) - \mathbf{Z}(sdt)) \\ & \geq f(\mathbf{p}^A(t)) - \mathcal{L}_{\mathbf{p}^A(t) - \mathbf{Z}(sdt)}(\mathbf{z}') \\ & = \sum_j \left(\int_{\tau \geq 0} \frac{w_j}{p_j} \tau z_j(\tau) d\tau - \int_{\tau \geq sdt} \frac{w_j}{p_j} (\tau - sdt) z_j(\tau) d\tau \right) \\ & \geq sdt \sum_j \frac{w_j}{p_j} \int_{\tau \geq sdt} z_j(\tau) d\tau = sdt(\tilde{W}^A(t) - \sum_j \frac{w_j}{p_j} Z_j(sdt)), \end{aligned}$$

which implies $\frac{d}{dt} f(\mathbf{p}^A(t)) \leq -s\tilde{W}^A(t)$.

Similarly, we can show the other direction. Consider any feasible s -speed processing \mathbf{Z}' during the time interval; so, j is processed by Z'_j units. Let $\mathbf{z} = \arg \min_{\mathbf{z}'} \mathcal{L}_{\mathbf{p}^A(t) - \mathbf{Z}'}(\mathbf{z}')$ denote the optimum residual schedule for the remaining job sizes. Let $\mathbf{Z}' \circ \mathbf{z}$ denote the 1-speed schedule where we first process \mathbf{Z}' for sdt time units and subsequently schedule \mathbf{z} . Note that $\mathbf{Z}' \circ \mathbf{z}$ is a feasible solution to $\mathcal{L}_{\mathbf{p}^A(t)}$. Therefore, $f(\mathbf{p}^A(t))$ decreases by

$$\begin{aligned} & f(\mathbf{p}^A(t)) - f(\mathbf{p}^A(t) - \mathbf{Z}') \\ & \leq \mathcal{L}_{\mathbf{p}^A(t)}(\mathbf{Z}' \circ \mathbf{z}) - \mathcal{L}_{\mathbf{p}^A(t) - \mathbf{Z}'}(\mathbf{z}) \\ & \leq \sum_j \left(\int_{\tau \geq 0} \frac{w_j}{p_j} (\tau + sdt) z_j(\tau) d\tau - \int_{\tau \geq 0} \frac{w_j}{p_j} \tau z_j(\tau) d\tau \right) + \sum_j sdt \frac{w_j}{p_j} Z'_j \\ & \leq sdt \sum_j \frac{w_j}{p_j} \int_{\tau \geq 0} z_j(\tau) d\tau + dt \sum_j \frac{w_j}{p_j} Z'_j \\ & = sdt\tilde{W}^A(t) + dt \sum_j \frac{w_j}{p_j} Z'_j, \end{aligned}$$

which implies $\frac{d}{dt} f(\mathbf{p}^A(t)) \geq -s\tilde{W}^A(t)$, as desired. \square

²¹In the proof of Theorem 3.2 we defined GD to be the most effective algorithm for decreasing the residual optimum, and we showed following the residual optimum schedule is as effective as GD. We make this small change, as we now know GD is really nothing but following the residual optimum.

This lemma allows us to bound the continuous change of the second term of $\Phi(t)$. The proof is essentially identical to that of Lemma 4.5: it follows from the observation that we can view A and O 's combined processing as a $s + 1$ -speed schedule and the residual objective can change at a rate up to the total fractional weight of jobs in the system, times $s + 1$.

Lemma 4.10. *Consider any time t when no job arrival or completion occurs. If A and O are given s -speed and 1-speed respectively, $\frac{d}{dt}f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t)) \geq -(s + 1)(\tilde{W}^A(t) + \tilde{W}^O(t))$.*

Combining the two lemmas we obtain the following corollary.

Corollary 4.11. *Consider any time t when no job arrival or completion occurs. When GD is given $1 + \epsilon$ -speed, we have $\frac{d}{dt}\Phi(t) \leq -\tilde{W}^A(t) + \frac{2+\epsilon}{\epsilon}\tilde{W}^O(t)$.*

By Lemma 4.8 and the fact that $\Phi(T) = \Phi(0) = 0$, it must be the case that $\int_{t=0}^{\infty} \frac{d}{dt}\Phi(t)dt \geq 0$. By Corollary 4.11 we know that $\int_{t=0}^{\infty} \tilde{W}^A(t)dt \leq \frac{2+\epsilon}{\epsilon} \int_{t=0}^{\infty} \tilde{W}^O(t)dt$ where the LHS and RHS are exactly the total weighted fractional flow time of GD and the optimum schedule (Proposition 2.3). Thus, we have Theorem 3.3 for total fractional weighted flow time. Lemma 2.1 allows us to obtain an algorithm that is $(1 + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive for the integer objective.

4.3 Proof of Theorem 3.4 This section is devoted to proving Theorem 3.4. The high-level idea is to show that when we take the feasible processing rate view, we satisfy the condition of Theorem 3.3, i.e., $v(\mathbf{x}) := \max_{0 \leq \mathbf{z} \leq \mathbf{x}, \mathbf{z} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{z}$ is linear substitute. Note that $\mathbf{z} \in \mathcal{P}$ if and only if $\mathbf{z}_j = u_j(\mathbf{y}_j)$ for all j for some non-negative vectors $\{\mathbf{y}_j\}_{j \in J}$ such that $\sum_j \mathbf{y}_j \leq 1$.

For any price vector \mathbf{q} , let $\mathbf{x}(\mathbf{q})$ denote the unique solution to:

$$\max_{\mathbf{x}'} \sum_{i \in J} (c_i - q_i)x'_i \\ \mathbf{x}' \in \mathcal{P}$$

To see that this is well defined, take the resource view of the CP. Recall in the resource view, each job i is associated with a valuation function u_i and the divisible resources $[0, 1]^d$ are allocated to the jobs. If \mathbf{y}_i denotes the resources job i receives, we have the following equivalent CP.

$$\max \sum_{i \in J} (c_i - q_i)u_i(\mathbf{y}_i) \\ \sum_{i \in J} \mathbf{y}_i \leq 1 \\ \mathbf{y}_i \geq \mathbf{0} \quad \forall i \in J \tag{4.9}$$

It is straightforward to see that this CP has a unique solution due to the theorem condition that u_i is strictly concave for all jobs i .

Without loss of generality, we assume $\mathbf{q} < \mathbf{c}$. This is because jobs i with $c_i - q_i \leq 0$ do not play any roles and therefore we can pretend that they do not exist. Now consider the Walrasian market where each job/agent i has a valuation function $(c_i - q_i)u_i$ for divisible resources $[0, 1]^d$. Because u_i is LS for all i , which is a precondition of Theorem 3.4, $(c_i - q_i)u_i$ is LS for all i (LS is closed under a positive scalar multiplication). Thus, the CP formulates the social surplus maximization problem where all agents have LS valuations. In the following \mathbf{b} and \mathbf{b}' are uniquely defined as u_i is assumed to be strictly monotone and strictly concave.

Lemma 4.12. *Consider any $\mathbf{q} \leq \mathbf{q}' \in \mathbb{R}^{|J|}$. Let \mathbf{b} be the equilibrium price vector for the market with valuations $(c_i - q_i)u_i$, $i \in J$ along with divisible resources $[0, 1]^d$. Similarly, let \mathbf{b}' be the equilibrium price vector for $(c_i - q'_i)u_i$, $i \in J$. We have $\mathbf{b}' \leq \mathbf{b}$.*

Proof. To prove this lemma, we use the tatonnement process described in [MS09].²² The tatonnement process is governed by a price vector $\hat{\mathbf{b}}(\tau)$; the vector monotonically changes as τ increases. We say that all resources are under-demanded²³ under the current price vector \mathbf{b}'' if there exists an allocation \mathbf{y}'' that maximizes

²²Although the process is described for indivisible resources that exist in multiple units, the same proof extends to divisible resources. The proof can be much simplified if we assume valuation functions are strictly concave, as shown in [AC04].

²³The over-demanded case is symmetric.

everyone's (here, every job i 's) quasi-linear utility and $\sum_i \mathbf{y}_i'' \leq 1$. Here, if job i 's utility is $(c_i - q_i)u_i$, then $\mathbf{y}_i'' \in \mathcal{D}((c_i - q_i)u_i, \mathbf{b}'')$. For brevity, we will equivalently say that \mathbf{b}'' is an under-demand price vector. It is shown in [MS09] that starting with any under-demand price vector $\hat{\mathbf{b}}(0)$, we can monotonically decrease $\hat{\mathbf{b}}(\tau)$ to an under-demand equilibrium price, which yields the maximum social surplus.

Let \mathbf{y} be the unique allocation achieving the unique equilibrium with the price vector \mathbf{b} . Define \mathbf{y}' analogously for $(c_i - q'_i)u_i$ with the price vector \mathbf{b} . Since for all i , $c_i - q_i \geq c_i - q'_i$ and $\mathbf{y}'_i \in \mathcal{D}((c_i - q'_i)u_i, \mathbf{b}) = \mathcal{D}((c_i - q_i)u_i, \frac{c_i - q_i}{c_i - q'_i} \mathbf{b})$, we have $\mathbf{y}'_i \leq \mathbf{y}_i$ due to the first property of Theorem 3.4. Thus, $\sum_i \mathbf{y}'_i \leq 1$, meaning \mathbf{b} is an under-demand price vector when each i has valuation $(c_i - q'_i)u_i$. Thanks to the above tatonnement process, we conclude $\mathbf{b}' \leq \mathbf{b}$. \square

Now we are ready to prove that $v(\mathbf{x})$ is LS. Say $q'_j > q_j$ and $q'_i = q_i$ for all $i \neq j$, so we only increased job j 's price. Consider any $i \neq j$. We use \mathbf{y} and \mathbf{y}' as we defined in the proof of Lemma 4.12. Since u_i is strictly concave, we have,

$$\begin{aligned} \{\mathbf{y}_i\} &= \mathcal{D}((c_i - q_i)u_i, \mathbf{b}) = \mathcal{D}(u_i, \mathbf{b}/(c_i - q_i)) \\ \{\mathbf{y}'_i\} &= \mathcal{D}((c_i - q'_i)u_i, \mathbf{b}') = \mathcal{D}((c_i - q_i)u_i, \mathbf{b}') = \mathcal{D}(u_i, \mathbf{b}'/(c_i - q_i)) \end{aligned}$$

From Lemma 4.12, we know $\mathbf{b}' \leq \mathbf{b}$. Thus, due to the second property stated in Theorem 3.4, we have $u_i(\mathbf{y}'_i) \geq u_i(\mathbf{y}_i)$. Knowing that $x_i(\mathbf{q}) = u_i(\mathbf{y}_i)$ and $x_i(\mathbf{q}') = u_i(\mathbf{y}'_i)$, we conclude

$$x_i(\mathbf{q}') \geq x_i(\mathbf{q}) \quad (4.10)$$

Lemma 4.13. *If $\mathbf{q} > \mathbf{0}$, then the demand set $\mathcal{D}(v, \mathbf{q})$ is a singleton set. Further, we have $\mathcal{D}(v, \mathbf{q}) = \{\mathbf{x}(\mathbf{q})\}$.*

Proof. The quasi-linear utility, $v(\mathbf{x}') - \mathbf{q} \cdot \mathbf{x}'$ is maximized only when $\mathbf{x}' \in \mathcal{P}$ since $\mathbf{q} > \mathbf{0}$. Then, the quasi-linear utility maximization can be written as $\max_{\mathbf{x}'} \mathbf{c} \cdot \mathbf{x}' - \mathbf{q} \cdot \mathbf{x}'$ s.t. $\mathbf{x}' \in \mathcal{P}$ which has a unique solution as discussed above. The solution is $\mathbf{x}(\mathbf{q})$ by definition. \square

Thus, if $\mathbf{q} > \mathbf{0}$, then we have $\mathbf{q}' > \mathbf{0}$ as well. Due to Lemma 4.13 and Eqn. (4.10), we have shown that v is LS. Therefore, we have Theorem 3.4 from Theorem 3.3.

We extend the proof to an arbitrary $\mathbf{q} \geq \mathbf{0}$. Let J^0 denote the set of jobs j with $q_j = 0$, and J^+ the other jobs. It is an easy exercise to show that

$$\mathcal{D}(v, \mathbf{q}) = \{\mathbf{x}(\mathbf{q}) + \sum_{j \in J^0} \lambda_j \mathbf{1}_j \mid \lambda_j \geq 0 \forall j \in J^0\}. \quad (4.11)$$

In other words, the quasi-linear utility is maximized by purchasing any amount of free jobs on top of $\mathbf{x}(\mathbf{q})$. Consider any $\mathbf{x} \in \mathcal{D}(v, \mathbf{q})$ and increase job j 's price, so we have \mathbf{q}' such that $q'_j > q_j$ and $q'_i = q_i$ for all $i \neq j$. For any $i \in J^0$ such that $i \neq j$, it is trivial to see that we can find $\mathbf{x}' \in \mathcal{D}(v, \mathbf{q}')$ such that $x'_i \geq x_i$ as job i is free. If $i \in J^+$, we have $x_i(\mathbf{q}') \geq x_i(\mathbf{q}) = x_i$, where the inequality follows from Eqn. (4.10) and the equality follows from Eqn. (4.11) with the facts that $\mathbf{x}, \mathbf{x}(\mathbf{q}) \in \mathcal{D}(v, \mathbf{q})$ and $i \in J^+$.

5 Applications of the Theorems

In this section, we discuss some problems captured by our theorems. See Section 1.3 for their definition.

5.1 Matroid Scheduling We can formulate the matroid scheduling problem as an instance of the PSP as follows:

$$\mathcal{P} = \{\mathbf{z} \mid \sum_{j \in S} z_j \leq r(S) \quad \forall S \subseteq J\}$$

where r is the rank function of the given matroid $\mathcal{M} = (J, \mathcal{I})$ and z_j is the processing rate of job j . For a cost vector \mathbf{c} , $\bar{v}(1_S) = \max_{\mathbf{z} \leq 1_S, \mathbf{z} \in 2^J, \mathbf{z} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{z}$ is equivalent to the weighted rank function of a matroid, which is known to be GS (Section 2.3). Note that $\bar{v}(1_S) = v(1_S)$ for all $S \subseteq J$ because the polytope \mathcal{P} is integral, where $v(\mathbf{x}) = \max_{\mathbf{z} \leq \mathbf{x}, \mathbf{z} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{z}$ (see Theorem 3.3).

To show v is LS, we prove that the concave closure of \bar{v} is equivalent to v . For the concave closure definition, see Theorem 2.15. Note that here we used \bar{v} to distinguish the discrete valuation from the continuous valuation v .

Claim 5.1. For $\mathbf{x} \in [0, 1]^J$, $v(\mathbf{x}) = \bar{v}^+(\mathbf{x})$.

Proof. First, we show the “ \leq ” direction. Let $\mathbf{z} \in \mathcal{P}$ such that $v(\mathbf{z}) = v(\mathbf{x})$. Since \mathcal{P} is integral, we can express \mathbf{z} as a convex combination of the subsets of J , so $\mathbf{z} = \sum_{S \in \mathcal{I}} \lambda_S \cdot \mathbf{1}_S$, where $\sum_{S \in \mathcal{I}} \lambda_S = 1$. Then, $v(\mathbf{x}) = \mathbf{c} \cdot \mathbf{z} = \sum_{S \in \mathcal{I}} \lambda_S (\mathbf{c} \cdot \mathbf{1}_S) = \sum_{S \in \mathcal{I}} \lambda_S \bar{v}(\mathbf{1}_S) \leq \bar{v}^+(\mathbf{x})$.

To show the other direction, let $\bar{v}^+(\mathbf{x}) = \sum_S \bar{v}(\mathbf{1}_S) \lambda_S$ for some $\{\lambda_S\}_{S \subseteq J}$ with $\sum_S \lambda_S = 1$ and $\sum_S \lambda_S \mathbf{1}_S = \mathbf{x}$. Let $\pi(S)$ be $S' \subseteq S$ such that $\mathbf{1}_{S'} \in \mathcal{P}$ and $\bar{v}(\mathbf{1}_{S'}) = \bar{v}(\mathbf{1}_S)$. We then have

$$\bar{v}^+(\mathbf{x}) = \sum_S \bar{v}(\mathbf{1}_{\pi(S)}) \lambda_S = \sum_S \lambda_S \mathbf{c} \cdot \mathbf{1}_{\pi(S)} \quad (5.12)$$

Let $\mathbf{x}' := \sum_S \lambda_S \mathbf{1}_{\pi(S)}$. Note that $\mathbf{x}' \leq \mathbf{x}$. Further, $\mathbf{x}' \in \mathcal{P}$ due to \mathcal{P} 's integrality. By definition of v , we know v is monotone. Thus, we have,

$$v(\mathbf{x}) \geq v(\mathbf{x}') = \mathbf{c} \cdot \mathbf{x}' = \sum_S \lambda_S \mathbf{c} \cdot \mathbf{1}_{\pi(S)} \quad (5.13)$$

From Eqn. (5.12) and (5.13), we conclude $v(\mathbf{x}) \geq \bar{v}^+(\mathbf{x})$, as desired. \square

Since \bar{v}^+ is LS, so is v . Therefore, Theorem 3.3 establishes GD as a scalable algorithm. Finally, if $\{\lambda_S\}_{S \in \mathcal{I}}$ is the convex combination of independent sets that achieves $v^+(\mathbf{x})$, we can implement our algorithm by preemptively scheduling each independent set S in proportional to λ_S .

5.2 Generalized Flow Scheduling Recall that we are given a network $G = (V, E)$, where $J \subseteq V$ is the set of source nodes. Arc e has capacity u_e and flow gain factor γ_e ; a flow of value f becomes $\gamma_e f$ after flowing through e . Job j gets processed at a rate equal to its (outgoing) net flow. Formally, node v has excess capacity b_v . Sources have non-negative excess capacities. A job j gets processed at a rate of z_j , when its net flow is z_j for $\mathbf{z} \in \mathcal{P}$, which is defined as follows.

$$\mathcal{P} = \left\{ \mathbf{z} = \left\langle \sum_w (f_{jw} - \gamma_{wj} f_{wj}) \right\rangle_{j \in J} \mid \sum_w (f_{vw} - \gamma_{vw} f_{wv}) \leq b_v \forall v \in V; f_e \leq u_e \forall e \in E; \mathbf{f} \geq 0 \right\} \quad (5.14)$$

Theorem 2.16 shows $v(\mathbf{x}) = \max_{\mathbf{z} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{z}$ is LS. Note that our network can ensure $\mathbf{z} \leq \mathbf{x}$ by setting the excess capacity of the node that represents job j to be x_j . Thus, we immediately have a scalable algorithm due to Theorem 3.3.

5.3 Speed-up Curves on Partitioned Resources We consider this problem from the Resource View. Then we need to show that the utility u_j is linear-substitute and satisfies two desired properties in Theorem 3.4. Without loss of generality, we can assume the utility is determined by only one univariate function that is $u_j(\mathbf{x}) = g(\sum_{d \in [D]} a_d x_d)$. We drop j for brevity henceforth.

We begin by showing a simple greedy algorithm being optimal. The proof follows from a simple swapping argument.

Lemma 5.2. For any price vector \mathbf{q} on resources, the greedy algorithm selects elements fractionally in decreasing order of a_d/q_d until no marginal increment, finds a solution maximizing the quasi-linear utility, i.e., a solution in $\mathcal{D}(\mathbf{u}, \mathbf{q})$.

Proof. Fix some price vector \mathbf{q} on resources. Assume that $a_1/q_1 \geq a_2/q_2 \geq \dots \geq a_D/q_D$. Suppose we have an optimum solution \mathbf{x} where $x_i < 1$ and $x_j > 0$ for some $i < j$. Then, we maximally increase x_i by $a_j \delta / a_i$ and decrease x_j by δ for some $\delta > 0$ until x_i becomes 1 or x_j becomes 0. This preserves the value of $\sum_d a_d x_d$ without increasing $\mathbf{q} \cdot \mathbf{x}$. By repeating this, we have the lemma. \square

First, we prove that u satisfies linear-substitute. Although the claim is known to be true (see Section 5 in [Mur22]), we include it as it is a nice warm-up before proving the subsequent claims.

Claim 5.3. u is linear-substitute.

Proof. Consider the greedy algorithm in Lemma 5.2. Now we increase the price of resource d . In the new greedy order, the resource d can only be considered later; other elements keep the same relative order. Thus \mathbf{x}_i^* can not decrease for all $i \neq d$ because they are processed earlier than the original greedy order. Therefore, we have the claim. \square

The following claim shows that when we increase the price vector \mathbf{q} simultaneously for each coordinate at the same scale, u can only demand less.

Claim 5.4. *For any $\mathbf{y} \in \mathcal{D}(u, \mathbf{q})$ and $\lambda \geq 1$, there exists $\mathbf{y}' \in \mathcal{D}(u, \lambda \mathbf{q})$ such that $\mathbf{y} \geq \mathbf{y}'$.*

Proof. Consider the greedy algorithm in Lemma 5.2. Note that increasing the price vector \mathbf{q} to $\lambda \mathbf{p}$ does not change the order of a_d/q_d . Thus the stopping point in the execution of the greedy algorithm comes earlier as the marginal increment becomes zero with fewer resources in the order, which gives the proof. \square

Further, we show that when we increase the price vector \mathbf{q} , the utility would not increase.

Claim 5.5. *For any $\mathbf{y} \in \mathcal{D}(u, \mathbf{q})$ and $\mathbf{q}' \geq \mathbf{q}$, there exists $\mathbf{y}' \in \mathcal{D}(u, \mathbf{q}')$ such that $u(\mathbf{y}) \geq u(\mathbf{y}')$.*

Proof. For the sake of contradiction, suppose $u(\mathbf{y}) < u(\mathbf{y}')$. Let \mathbf{y}' be the solution constructed by the greedy algorithm described in the proof of Lemma 5.2 with price vector \mathbf{q}' . Define \mathbf{y} analogously with \mathbf{q} . Because of the assumption $u(\mathbf{y}) < u(\mathbf{y}')$ and g 's monotonicity, it must be the case that $\sum_i a_i y_i < \sum_i a_i y'_i$. Let \mathbf{y}'' be the “prefix” of \mathbf{y}' such that $z := \sum_i a_i y_i = \sum_i a_i y''_i$. In other words, we run the greedy with \mathbf{q}' but stop when $z = \sum_i a_i y''_i$. Let k be the smallest such that $y'_k > y''_k$. The greedy with respect to \mathbf{q}' further increases \mathbf{y}'' since the marginal gain $g(z + \epsilon a_k) - g(z) - \epsilon q'_k > 0$ for a sufficiently small $\epsilon > 0$. Therefore, if $\mathbf{y}_k < 1$, the greedy must have continued increasing \mathbf{y} since $g(z + \epsilon a_k) - g(z) - \epsilon q_k \geq g(z + \epsilon a_k) - g(z) - \epsilon q'_k > 0$. Since the greedy with respect to \mathbf{q} considers resources in the order of $1, 2, \dots$, it means $\mathbf{y}_1 = \mathbf{y}_2 = \dots = \mathbf{y}_k = 1$; yet $\mathbf{y}''_k < 1$ and $\mathbf{y}''_{k+1} = \mathbf{y}''_{k+2} = \dots = \mathbf{y}''_D = 0$. This is a contradiction to $\sum_i a_i y_i < \sum_i a_i y'_i$. \square

Finally, we discuss why it is wlog to assume that u is strictly monotone and strictly concave. To obtain a strictly concave function that is arbitrarily close to u , we replace each x_d with $x_d^{1-\delta}$ for an arbitrary small $\delta > 0$. Further, we replace g with a strictly monotone function that is arbitrarily close to g . It is an easy exercise to show this new function is strictly concave and arbitrarily close to the original function u on $[0, 1]^D$. We then pretend an infinitesimal unit ϵ of each resource as a distinct item and order these infinitesimal-sized items in decreasing order of $a_d((k\epsilon)^{1-\delta} - ((k-1)\epsilon)^{1-\delta})/(p_d\epsilon)$, which corresponds to the effectiveness of the k -th ϵ -sized copy of resource d . It is straightforward to check that the whole analysis remains unchanged. Further, if u does not depend on some resources, then for each of such resources d we can add $\lambda_d x_d^{1-\delta}$ for a sufficiently small $\lambda_d > 0$, so this additional term essentially has no effect on u while ensuring u being strictly monotone.

Thus, this application admits a scalable algorithm due to Theorem 3.4.

5.4 Restricted Assignment for Unweighted Jobs In this section we show how to obtain a $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive algorithm for total unweighted flow time in the restricted assignment model using gradient descent. Our algorithm is migratory, unlike the algorithms in [CGKM09, AGK12]. (We also reproduce the non-migratory result in [AGK12] in Section 5.7, focusing on supermodularity and gradient descent.) In the restricted assignment case, we have $\lambda_{ij} \in \{0, 1\}$ for all i and j .

This section acts as a warm-up for unweighted unrelated machines. Unfortunately, for unrelated machines we can not directly use Theorem 3.2. We assume migration of jobs to be allowed across machines. The residual optimum of a migratory schedule does not seem to be LS. Thus, we use the non-migratory residual optimum, which turns out to be GS. Interestingly, our resulting gradient descent algorithm is migratory. Nevertheless, the analysis will be very similar to the proof of Theorem 3.2. Thus, we will adopt the same notation throughout this section.

We first describe the residual optimum for this problem. Let $f(\mathbf{x})$ denote (the cost of) the min-cost non-migratory schedule of jobs, where each job j has (remaining) size x_j . It is well known that it can be computed by min-cost bipartite matching. For completeness, we describe it here: Consider a bipartite graph $G = (J, M'; E)$ where there is a unique node, indexed by (i, k) , for each pair of machine i and $k \in \mathbb{Z}_{>0}$. There is an edge between $j \in J$ and $(i, k) \in M'$. Choosing the edge means scheduling job j on machine i in the k -th reverse order non-preemptively, thus the edge has cost $k \cdot p_{ij}$, where $p_{ij} = x_j/\lambda_{ij}$. Intuitively, this means there are k jobs behind job

j on the machine, including itself—thus j delays k jobs—when following SRPT on each machine. The residual optimum $f(\mathbf{x})$ is the min cost of a bipartite matching for this instance.

Lemma 5.6. $\bar{g}_{\mathbf{x}}(J') := f(\mathbf{x} \odot \mathbf{1}_{J'})$ is (discrete)-supermodular.

Proof. From Section 2.3.1, we know that maximum weight bipartite matching is GS. Therefore, $-\bar{g}_{\mathbf{p}}$ is GS and submodular. \square

We use $\text{GD}(\mathbf{x})$ to denote the decrease rate of the residual optimum $f(\mathbf{x})$ when jobs are processed on each machine according to SRPT without migrating jobs across machines and without speed augmentation. We recompute the residual optimum when a new job arrives or GD completes a job.

Since the analysis is very similar to the proof of Theorem 3.2, we only highlight the differences. We use the same potential function. Lemma 5.6 gives supermodularity of the residual optimum. Thus, we can show all discrete changes of Φ are non-positive following the same argument in the proof of Lemma 4.4.

To make the analysis more transparent, we will assume wlog that any fixed algorithm processes at most one job per machine at a time. This is because a feasible schedule can be expressed as a fractional matching between jobs and machines and can be decomposed into a convex combination of integral matchings.

Lemma 5.7. When the residual optimum is $f(\mathbf{x})$, we have $\sum_{x_j > 0} 1 \leq |\text{GD}(\mathbf{x})|$. Further, $|B(\mathbf{x})| \leq E^B(\mathbf{x}) + \sum_{x_j > 0} 1$, where $B(\mathbf{x})$ denotes the decrease rate of $f(\mathbf{x})$ for an arbitrary algorithm B and $E^B(\mathbf{x})$ denotes the number of jobs processed by B at the moment.

Proof. Proving $\text{GD}(\mathbf{x}) \leq -\sum_{x_j > 0} 1$ is identical to the first part proof of Lemma 4.2. In fact, we can decrease $f(\mathbf{x})$ at a rate of at least $\sum_{x_j > 0} 1$ by following the residual optimum schedule.

Showing the other inequality needs special care because while the residual optimum schedule is non-migratory, changing the job assignment could help decrease the residual optimum. Assume wlog that $\mathbf{x} > 0$, because we can ignore jobs j such that $x_j = 0$. Suppose B processes $j(i)$ on machine i for a sufficiently small unit time dt , so no job completes during the time slot. Let \mathbf{x}' be the resulting remaining job sizes and σ' be the optimal non-migratory schedule for \mathbf{x}' , where j is scheduled on machine $\psi(j)$. Let σ be the non-migratory schedule obtained from σ' by increasing each job's remaining size from x'_j to x_j , without changing the assignment ψ and the job ordering on each machine. So, σ is a feasible non-migratory schedule for \mathbf{x} , but not necessarily optimum. Let $\text{cost}(\sigma)$ denote the total completion time of schedule σ . Clearly we have,

$$B(\mathbf{x}) \cdot dt \geq \text{cost}(\sigma') - \text{cost}(\sigma).$$

Thus, it suffices to lower bound $\text{cost}(\sigma') - \text{cost}(\sigma)$. For notational brevity, let $x'_{ij} := x'_j / \lambda_{ij}$, which is j 's processing time on i when its remaining size is x'_j . Similarly, let $x_{ij} := x_j / \lambda_{ij}$. Define $P_{i, < k} := \sum_{j': x'_{ij'} < x'_{ik}} x'_{ij'}$ to be the total remaining processing time of jobs smaller than job k on machine i w.r.t. \mathbf{x}' in σ' , breaking ties in an arbitrary but fixed order. Let $N_{i, \geq k} := \sum_{j': x'_{ij'} \geq x'_{ik}} 1$ denote the number of jobs of remaining sizes no smaller than job k on machine i w.r.t. \mathbf{x}' in σ' .

Since $j(i)$ gets processed on machine i , we have $x_{j(i)} - x'_{j(i)} = \lambda_{i, j(i)} dt$. The job $j(i)$ is on machine $\psi(j(i))$, and the number of jobs behind it²⁴, including itself, is exactly $N_{\psi(j(i)), \geq j(i)}$. Thus, due to $x'_{j(i)}$ increasing to $x_{j(i)}$, $j(i)$'s contribution to $\text{cost}(\sigma') - \text{cost}(\sigma)$ is exactly,

$$N_{\psi(j(i)), \geq j(i)} x'_{\psi(j(i)), j(i)} - N_{\psi(j(i)), \geq j(i)} x_{\psi(j(i)), j(i)} = -N_{\psi(j(i)), \geq j(i)} \frac{\lambda_{i, j(i)}}{\lambda_{\psi(j(i)), j(i)}} dt \quad (5.15)$$

Therefore, we have

$$B(\mathbf{x}) \geq - \sum_i N_{\psi(j(i)), \geq j(i)} \frac{\lambda_{i, j(i)}}{\lambda_{\psi(j(i)), j(i)}} dt$$

We complete the proof by showing

$$N_i + 1 \geq N_{\psi(j(i)), \geq j(i)} \frac{\lambda_{i, j(i)}}{\lambda_{\psi(j(i)), j(i)}} \quad (5.16)$$

²⁴Note that on each machine i jobs are ordered following the SRPT order, i.e. in decreasing order of x_{ij} .

for all i , where $N_i := |\psi^{-1}(i)|$ is the number of jobs assigned to machine i . To see this, observe the following:

$$P_{i, < j(i)} + N_{i, \geq j(i)} \frac{x'_{j(i)}}{\lambda_{i, j(i)}} + \frac{x'_{j(i)}}{\lambda_{i, j(i)}} \geq P_{\psi(j(i)), < j(i)} + N_{\psi(j(i)), \geq j(i)} \frac{x'_{j(i)}}{\lambda_{\psi(j(i)), j(i)}}, \quad (5.17)$$

This is because the LHS and RHS measure the increase of the objective due to $j(i)$'s presence on machines i and $\psi(j(i))$, respectively (in other words, the LHS is the increase of the residual objective on machine i when we move $j(i)$ from $\psi(j(i))$ to i , and the RHS is the decrease of the residual objective on machine $\psi(j(i))$ for that move): The first term is $j(i)$'s start time and the second is the number of jobs delayed by $j(i)$, including itself, multiplied by $j(i)$'s remaining size on the machine. Thus, the inequality is a necessary condition for σ' being the optimal residual schedule for \mathbf{x}' .

Multiplying both sides in Eqn. (5.17) by $\frac{\lambda_{i, j(i)}}{x'_{j(i)}} = \frac{1}{x'_{i, j(i)}}$, we have

$$\frac{P_{i, < j(i)}}{x'_{i, j(i)}} + N_{i, \geq j(i)} + 1 \geq \frac{P_{\psi(j(i)), < j(i)}}{x'_j / \lambda_{i, j(i)}} + N_{\psi(j(i)), \geq j(i)} \frac{\lambda_{i, j(i)}}{\lambda_{\psi(j(i)), j(i)}}. \quad (5.18)$$

For the first term, we have $\frac{P_{i, < j(i)}}{x'_{i, j(i)}} = \sum_{j': x'_{ij'} < x'_{i, j(i)}} x'_{ij'} / x'_{i, j(i)} \leq \sum_{j': x'_{ij'} < x'_{i, j(i)}} 1$. Thus, it is at most the number of jobs that have smaller processing times than $j(i)$ on machine i . Thus, the LHS is at most $N_i + 1$ per machine. Finally, taking a summation of all machines that process jobs, we have the lemma. \square

Let $E^A(t)$ and $E^O(t)$ denote the number of jobs processed by A and O at time t respectively. We have a lemma similar to Lemma 4.5. However, we need to take care of extra terms E^A and E^O due to the processing of jobs in A and O . Let $N^A(t) := |A(t)|$ and $N^O(t) := |O(t)|$.

Lemma 5.8. *Consider any time t when no job arrival or completion occurs. If A and O are given s -speed and 1-speed respectively, $\frac{d}{dt} f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t)) \geq -(s+1)(N^A(t) + N^O(t)) - (sE^A(t) + E^O(t))$.*

Proof. Similar to the proof of Lemma 4.5, we can think of processing jobs in A and O as a feasible schedule with speed $s+1$. More precisely, jobs in $A(t)$ are processed with s speed and jobs in $O(t)$ are processed with 1 speed. Thanks to Lemma 5.7 GD's processing can change the residual at a rate of at most $s(N^A(t) + N^O(t) + E^A(t))$. Similarly, the optimum solution's processing can change residual at a rate of at most $N^A(t) + N^O(t) + E^O(t)$. The lemma then follows immediately. \square

This lemma allows us to bound the continuous change of $\Phi(t)$.

Corollary 5.9. *Consider any time t when no job arrival or completion occurs. Then, $\frac{d}{dt} \Phi(t) \leq -N^A(t) + O(1/\epsilon)(E^A(t) + E^O(t) + N^O(t))$ when GD is given $s = (1 + \epsilon)$ -speed.*

Proof. We have $\frac{d}{dt} \Phi(t) = \frac{2}{\epsilon} \left(-sN^A(t) + \frac{s+1}{2}(N^A(t) + N^O(t)) + \frac{1}{2}(sE^A(t) + E^O(t)) \right) \leq -N^A(t) + O(1/\epsilon)(E^A(t) + E^O(t) + N^O(t))$. \square

Since all discrete changes in the potential are non-positive, we have $\frac{d}{dt} \Phi(t) \geq 0$. From Corollary 5.12, we have $\int_{t=0}^{\infty} N^A(t) dt \leq O(1/\epsilon) \int_{t=0}^{\infty} (E^A(t) + E^O(t) + N^O(t)) dt$. In the restricted assignment model, where $\lambda_{ij} \in \{0, 1\}$, we have $\int_{t=0}^{\infty} E^A(t) dt = \int_{t=0}^{\infty} E^O(t) dt = \sum_j p_j$, which is a clear lower bound on the optimum. Therefore, we have an algorithm that is $(1 + \epsilon)$ -speed $O(1/\epsilon)$ competitive for the integral objective. All the remaining analysis remains the same and is omitted.

5.5 Unrelated Machines for Unweighted Jobs The restricted assignment scheduling analysis in Section 5.4 can be easily extended to the case for unrelated machines with minor modifications. First, notice that the analysis for the restricted assignment model directly would not give a competitive guarantee with $1 + \epsilon$ -speed for the general unweighted case. This is because while $\int_{t=0}^{\infty} \lambda \cdot E^A(t) dt$ is a valid lower bound on the optimum in the restricted assignment, but it is not the case in general. To address this issue, we aim to cancel off the positive terms of $E^A(t)$ in $\frac{d}{dt} \Phi(t)$. Towards this end, we put more weight on processing jobs to obtain more credits from processing individual jobs compared to reducing the inter-job delays. Thus, instead of ordering jobs from the 1-st reverse

order, we conceptually order jobs from the $1/\epsilon + 1$ -th reverse order. Formally, we define the residual optimum as follows. Consider a bipartite graph $G = (J, M'; E)$ where there is a unique node, indexed by $(i, k + \frac{1}{\epsilon})$, for each pair of machine i and $k \in \mathbb{Z}_{>0}$. There is an edge between $j \in J$ and $(i, k) \in M'$. Choosing the edge means scheduling job j on machine i in the k -th reverse order non-preemptively, thus the edge has cost $(k + \frac{1}{\epsilon}) \cdot p_{ij}$, where $p_{ij} = x_j/\lambda_{ij}$. The residual optimum $f(\mathbf{x})$ is the min cost of a bipartite matching for this instance.

Due to the change in the order, Lemma 5.7 needs to be modified.

Lemma 5.10. *When the residual optimum is $f(\mathbf{x})$, we have $\frac{1}{\epsilon}E^{\text{GD}}(\mathbf{x}) + \sum_{x_j > 0} 1 \leq |\text{GD}(\mathbf{x})|$. Further, $|B(\mathbf{x})| \leq (1 + \frac{1}{\epsilon})E^B(\mathbf{x}) + \sum_{x_j > 0} 1$, where $B(\mathbf{x})$ denotes the decrease rate of $f(\mathbf{x})$ for an arbitrary algorithm B and $E^B(\mathbf{x})$ denotes the number of jobs processed by B at the moment.*

The proof is essentially the same as Lemma 5.7, with the only change being when a job is processed on a machine, instead of getting the “+1” term per machine, we get $1 + \frac{1}{\epsilon}$ due to the reverse ranks being increased by $1/\epsilon$. This is an easy exercise to check. Therefore, similarly, we have

Lemma 5.11. *Consider any time t when no job arrival or completion occurs. If A and O are given s -speed and 1-speed respectively, $\frac{d}{dt}f(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t)) \geq -(s+1)(N^A(t) + N^O(t)) - (1 + \frac{1}{\epsilon})(sE^A(t) + E^O(t))$.*

Corollary 5.12. *Consider any time t when no job arrival or completion occurs. Then, $\frac{d}{dt}\Phi(t) \leq -\Omega(1/\epsilon)E^A(t) - \Omega(1)N^A(t) + O(1/\epsilon^2)(E^O(t) + N^O(t))$ when GD is given $s = (1 + 10\epsilon)$ -speed for $\epsilon < 1/10$.*

All remaining analysis remains the same as the restricted assignment case and we obtain an algorithm that is $(1 + 10\epsilon)$ -speed $O(1/\epsilon^2)$ competitive for the integral objective.

5.6 Unrelated Machines for Weighted Jobs In this section, we extend the GD algorithm to handle weighted jobs and give an algorithm that is $(1 + \epsilon)$ -speed $O(1/\epsilon^3)$ -competitive algorithm for total weighted flow time. One may try the previous residual LP (Eqn. (3.4)) where

$$\mathcal{P} = \left\{ z_j(\tilde{t}) = \sum_i \lambda_{ij} y_{ij}(\tilde{t}) \mid \sum_i y_{ij}(\tilde{t}) \leq 1 \ \forall j, \tilde{t}; \sum_j y_{ij}(\tilde{t}) \leq 1 \ \forall i, \tilde{t} \right\}$$

Unfortunately, this objective does not seem to be LS. Instead, we consider essentially the same LP that was used in [AGK12] for their dual fitting analysis. However, our algorithm is very different from the immediate-dispatch algorithm that was given in [AGK12]. For notional simplicity, we may use $p_{ij} := p_j/\lambda_{ij}$ to denote the processing time of job j on machine i when j is processed on machine i only.

$$\begin{aligned} \text{Residual LP } \mathcal{L}_{\mathbf{x}} : \quad f(\mathbf{x}) := \min & \underbrace{\sum_{i,j} \frac{w_j \lambda_{ij}}{p_j} \int_{\tilde{t} \geq 0} \tilde{t} \cdot z_{ij}(\tilde{t}) d\tilde{t}}_{(*)} + \underbrace{\sum_{i,j} w_j \int_{\tilde{t} \geq 0} z_{ij}(\tilde{t}) d\tilde{t}}_{(**)} \\ & \sum_j z_{ij}(\tilde{t}) \leq 1 \quad \forall i, \tilde{t} \\ & \sum_i \int_{\tilde{t} \geq 0} \lambda_{ij} z_{ij}(\tilde{t}) d\tilde{t} = x_j \quad \forall j \end{aligned} \tag{5.19}$$

This new objective will be shown to be supermodular from the LS property (see Lemma 5.15). Notice that this LP does not force a job to be processed on at most one machine; thus $\mathbf{z}(\tilde{t})$ may not be a feasible schedule. For this reason, we can not directly apply the GD algorithm in Section 3.2, which schedules $\mathbf{z}(\tilde{t} = 0)$.

To handle this issue, we consider the following *approximate* GD: For each machine i , we only process one unit of jobs that appear the earliest in $\mathbf{z}(\tilde{t})$, ensuring that each job j gets processed up to the fraction of the job assigned to the machine. Formally, let \tilde{t}_i be the earliest time that $\sum_j \int_{\tilde{t}=0}^{\tilde{t}_i} \frac{z_{ij}(\tilde{t})}{p_{ij}} d\tilde{t} = 1$. Then we process $\hat{z}_{ij} := \int_{\tilde{t}=0}^{\tilde{t}_i} \frac{z_{ij}(\tilde{t})}{p_{ij}} d\tilde{t}$ fraction of job j on machine i . We continue this schedule until the fraction of a job on some machine is completed. For brevity, we may call this approximate GD as GD.

Remark 5.13. If we use the true GD algorithm, which decreases the residual optimum objective the most, we can still show the same guarantee. However, for the sake of analysis, we will consider the approximate GD.

We first verify that $\hat{\mathbf{z}}$ is a feasible schedule. By definition of the algorithm, we have $\sum_j \hat{z}_{ij} \leq 1$ for all i . This, together with the following claim, shows that \hat{z}_{ij} is a fractional matching between jobs and machines. Since a fractional bipartite matching can be expressed as a convex combination of integral matchings, we have that \hat{z}_{ij} is indeed a feasible schedule.

Claim 5.14. For any job j , we have $\sum_i \hat{z}_{ij} \leq 1$.

Proof. From the second constraint of LP, we have

$$\sum_i \hat{z}_{ij} = \sum_i \int_{\tilde{t}=0}^{\tilde{t}_i} \frac{z_{ij}(\tilde{t})}{p_{ij}} d\tilde{t} \leq \sum_i \int_{\tilde{t}=0}^{\tilde{t}_i} \frac{\lambda_{ij} z_{ij}(\tilde{t})}{p_j} d\tilde{t} = \frac{x_j}{p_j} \leq 1.$$

□

For simplicity, we assume wlog that $\sum_i \hat{z}_{ij} = 1$ and $\sum_j \hat{z}_{ij} = 1$, by adding 0 weight jobs.

For the analysis, we use the following potential function.

$$\Phi(t) = \frac{2}{\epsilon} \left(f(\mathbf{p}^A(t)) - \epsilon f(\mathbf{p}^A(t) \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O(t)) \right)$$

Here we note that $f(\mathbf{p}^A(t) \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O(t))$ is the optimum objective of \mathcal{L} when we have one copy of each job j in $A(t)$ with remaining size $p_j^A(t)$ and $(1/\epsilon - 1)$ copies of each job j in $O(t)$, each with remaining size $p_j^O(t)$.

5.6.1 Analysis Overview Before diving into the analysis, we first provide a high-level overview. Since the LP solution does not directly give us a time-indexed feasible schedule, we compromised true gradient descent. If possible, it would be the best for the algorithm to process exactly what the LP solution processes at time $\tilde{t} = 0$. Unfortunately as discussed before, this may not be feasible.

To intuitively understand our algorithm, consider a single machine i where jobs $1, 2, 3, \dots$ are processed in this order, each by a half fraction. For simplicity, let us drop i from the notation. Then, the LP solution on the machine is processing j for $p_j/2$ time steps. As mentioned before, fully processing job 1 on the machine may result in an infeasible schedule if job 1 is scheduled before other jobs on another machine in the LP solution. Our algorithm processes job 1 and job 2, each by half on the machine (these two jobs are the first unit fraction of jobs), to ensure a feasible schedule. This is surely not so effective as fully processing job 1 in decreasing (*). However, for all jobs except 1 and 2, our algorithm is equally effective! Further, for the deficit due to jobs 1 and 2, (**) comes to rescue. So, if $\tilde{W}_{(1)}$ is the total fractional weight processed by our algorithm, and we let $\tilde{W}_{(2)}$ be the other total fractional weight, our algorithm gets $\tilde{W}_{(1)}$ credit from (**) and $\tilde{W}_{(2)}$ credit from (*). Thus, $f(\mathbf{p}^A(t))$ decreases at a rate of at least $\tilde{W}_{(1)} + \tilde{W}_{(2)}$, which is the total fractional weight of the remaining jobs in our algorithm's schedule.

To show the algorithm is scalable, we create $(1/\epsilon - 1)$ copies of each job in the adversary's schedule and due to this we lose another $1/\epsilon$ factor in the competitive ratio, but this is a minor technicality. The critical observation is that the approximate GD is as effective as GD with the aid of the decrease of (**).

As before we will consider the discrete changes and continuous changes separately.

5.6.2 Analysis: Discrete Changes The proof of discrete changes of Φ is very similar to Lemma 4.7, so we only highlight the differences. First, we will use generalized flow to show the supermodularity of our new residual objective.

Lemma 5.15. $f(x)$ is (discrete-)supermodular.

Proof. We follow the same strategy as we used to prove Lemma 4.7. Consider the following valuation

$$v_{\tilde{t}}(\mathbf{x}(\tilde{t})) := \max_{i,j} \sum \left(B\lambda_{ij} - \frac{w_j \tilde{t}}{p_{ij}} - 1 \right) z_{ij} \text{ s.t. } 0 \leq \sum_i \lambda_{ij} z_{ij} \leq x_j(\tilde{t}), \sum_j z_{ij} \leq 1 \quad (5.20)$$

for each time \tilde{t} . It is easy to see that the LP objective by flipping the sign and adding a big constant is equivalent to

$$\max \int_{\tilde{t} \geq 0} v_{\tilde{t}}(\mathbf{x}(\tilde{t})) d\tilde{t} \text{ s.t. } \int_{\tilde{t} \geq 0} \mathbf{x}(\tilde{t}) d\tilde{t} \leq \mathbf{x}$$

To show $f(x)$ is supermodular, it is sufficient to show that $v_{\tilde{t}}$ is LS for each time \tilde{t} . Toward this end, we can interpret the constraints in (5.20) in the generalized flow setting as follows. For each machine i , node i has excess capacity 1. For each job j , node j has excess capacity $x_j(\tilde{t})$. Then add an arc from j to i with gain factor $1/\lambda_{ij}$ for each job j and machine i . Thus the constraints fall in the generalized flow polytope (5.14). Then we let the cost from j to i be $(B\lambda_{ij} - \frac{w_j \tilde{t}}{p_{ij}} - 1)$. From Theorem 2.16, we know $v_{\tilde{t}}$ is LS which leads to the lemma. \square

For the sake of completeness, we give the proof of the following lemma that bounds the change of jobs' arrival or completion.

Lemma 5.16. *The change of $\Phi(t)$ due to the job's completion or arrival is non-positive.*

Proof. The proof is almost identical to Lemma 4.4 although we use a slightly different potential function. Thus we only point out the differences when a new job j arrives. Then $f(\mathbf{p}^A \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O)$ increases by at least:

$$\begin{aligned} & f(\mathbf{p}^A \parallel p_j \mathbf{1}_{j^A} \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O \parallel (\frac{1}{\epsilon} - 1)p_j \mathbf{1}_{j^O}) - f(\mathbf{p}^A \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O) \\ &= g_{\mathbf{y}}(A \cup j^A \cup (\frac{1}{\epsilon} - 1)O \cup (\frac{1}{\epsilon} - 1)j^O) - g_{\mathbf{y}}(A \cup (\frac{1}{\epsilon} - 1)O) \end{aligned}$$

where $\mathbf{y} = \mathbf{p}^A \parallel p_j \mathbf{1}_{j^A} \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O \parallel (\frac{1}{\epsilon} - 1)p_j \mathbf{1}_{j^O}$. By supermodularity of g and using the fact that $p^A(r_j) = p^O(r_j) = p_j$, we have

$$\begin{aligned} & g_{\mathbf{y}}(A \cup j^A \cup (\frac{1}{\epsilon} - 1)O \cup (\frac{1}{\epsilon} - 1)j^O) - g_{\mathbf{y}}(A \cup (\frac{1}{\epsilon} - 1)O) \\ & \geq g_{\mathbf{y}}(A \cup j^A \cup (\frac{1}{\epsilon} - 1)j^O) - g_{\mathbf{y}}(A) \\ & = g_{\mathbf{y}}(A \cup (1/\epsilon)j^A) - g_{\mathbf{y}}(A) \end{aligned}$$

Then we can decompose it to a telescopic sum and apply the supermodularity as follows.

$$\begin{aligned} &= \left(g_{\mathbf{y}}(A \cup (1/\epsilon)j^A) - g_{\mathbf{y}}(A \cup (1/\epsilon - 1)j^A) \right) + \dots + \left(g_{\mathbf{y}}(A \cup j^A) - g_{\mathbf{y}}(A) \right) \\ & \geq \frac{1}{\epsilon} (g_{\mathbf{y}}(A \cup j^A) - g_{\mathbf{y}}(A)) \end{aligned}$$

Therefore, $\Phi(t)$'s increase is at most $\frac{2}{\epsilon} (g_{\mathbf{y}}(A \cup j^A) - g_{\mathbf{y}}(A)) - \epsilon \cdot \frac{1}{\epsilon} (g_{\mathbf{y}}(A \cup j^A) - g_{\mathbf{y}}(A)) = 0$, as desired. \square

5.6.3 Analysis: Continuous Changes As before, we let $\tilde{W}^A(t) = \sum_j \frac{w_j}{p_j} p_j^A(t)$ be the total remaining fractional weight of our algorithm A at time t . Then, let $\tilde{W}_{(1)}^A(t)$ be the fractional weight of jobs processed by the algorithm at t , i.e., $\tilde{W}_{(1)}^A(t) := \sum_{i,j} w_j \int_{\tilde{t}=0}^{\tilde{t}_i} \frac{z_{ij}(\tilde{t})}{p_{ij}} d\tilde{t} = \sum_{i,j} w_j \hat{z}_{ij}$. Let $\tilde{W}_{(2)}^A(t) := \tilde{W}^A(t) - \tilde{W}_{(1)}^A(t)$. For brevity, in all of the following lemmas, we omit that we consider a time t when there is no jobs' completion and arrival. The following lemma shows that the approximate GD still decreases the new residual optimum effectively.

Lemma 5.17. $\frac{d}{dt} f(\mathbf{p}^A(t)) \leq -s \tilde{W}^A(t)$.

Proof. Fix any time t and consider the change during time $[t, t + dt)$. Let \mathbf{z} be the optimum solution of $f(\mathbf{p}^A(t))$, i.e., $\mathbf{z} = \arg \min_{\mathbf{z}'} \mathcal{L}_{\mathbf{p}^A(t)}(\mathbf{z}')$. Recall that our algorithm schedules $\hat{\mathbf{z}}$ during time interval $[t, t + dt)$ and it is given s -speed, i.e., GD processes a job j on machine i by $sdt \cdot \hat{z}_{ij}$ fraction, where $\hat{z}_{ij} := \int_{\tilde{t}=0}^{\tilde{t}_i} \frac{z_{ij}(\tilde{t})}{p_{ij}} d\tilde{t}$. Now following the same strategy in Lemma 4.9, we need to find a feasible schedule \mathbf{z}' (with 1-speed) for remaining job sizes $\mathbf{p}^A(t) - sdt \hat{\mathbf{Z}}$ where $\hat{\mathbf{Z}}_j := \sum_i \lambda_{ij} \hat{z}_{ij}$. Then, the construction of \mathbf{z}' is as follows. First, we remove $sdt \cdot z_{ij}(\tilde{t})/p_{ij}$

fraction from \mathbf{z} at local time $\tilde{t} \leq \tilde{t}_i$ for job j on machine i . Note that this empties out sdt units of space on the machine because $\sum_j \int_{\tilde{t}=0}^{\tilde{t}_i} \frac{z_{ij}(\tilde{t})}{p_{ij}} d\tilde{t} = 1$. We obtain \mathbf{z}' by removing the empty spaces on each machine.

Let's fix machine i and consider the change of $(*)$ in the residual LP (5.19). For any $\tilde{t} > \tilde{t}_i$, since $sdt \cdot \sum_j \hat{z}_{ij} = sdt$ fraction of jobs completed before time \tilde{t}_i , the contribution of $z_{ij}(\tilde{t})$ in $\mathcal{L}_{\mathbf{p}^A(t+dt)}(\mathbf{z}')$ becomes at most $\frac{w_j}{p_{ij}}(\tilde{t} - sdt)z_{ij}(\tilde{t})$.

Thus, the decrease of $f(\mathbf{p}^A(t))$ from $(*)$ is by at least

$$\sum_{i,j} \frac{w_j}{p_{ij}} \int_{\tilde{t} \geq \tilde{t}_i} (\tilde{t} - (\tilde{t} - sdt)) z_{ij}(\tilde{t}) d\tilde{t} = sdt \cdot \sum_{i,j} \frac{w_j}{p_{ij}} \int_{\tilde{t} \geq \tilde{t}_i} z_{ij}(\tilde{t}) d\tilde{t} = sdt \cdot \tilde{W}_{(2)}^A(t)$$

Since job j gets processed by $sdt \cdot \hat{z}_{ij}$ fraction on machine i , the change of $f(\mathbf{p}^A(t))$ from $(**)$ is exactly

$$\sum_{i,j} w_j \left(\int_{\tilde{t} \geq 0} z_{ij}(\tilde{t}) d\tilde{t} - \int_{\tilde{t} \geq 0} z'_{ij}(\tilde{t}) d\tilde{t} \right) = \sum_{i,j} w_j \cdot sdt \cdot \hat{z}_{ij} = sdt \cdot \tilde{W}_{(1)}^A(t)$$

Therefore, the lemma follows by putting two bounds together. \square

In the following $|_{GD}$ and $|_{\text{opt}}$ mean that we consider the change due to GD's processing and opt 's processing alone respectively.

Lemma 5.18. $-\frac{d}{dt} f(\mathbf{p}^A(t) \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O(t)) \Big|_{GD} \leq 2s\tilde{W}^A(t) + s(\frac{1}{\epsilon} - 1)\tilde{W}^O(t)$.

Proof. Fix a time t and consider the change during $[t, t + dt)$, where no job arrives or completes assuming that only GD processes jobs. For notational simplicity, let $\mathbf{x}(t) := \mathbf{p}^A(t) \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O(t)$. To achieve our goal of upper bounding the following quantity,

$$f(\mathbf{x}(t)) - f(\mathbf{x}(t + dt)),$$

we construct a feasible solution \mathbf{z}'' to $\mathcal{L}_{\mathbf{x}(t)}$. Let \mathbf{z}' be an optimum solution of $f(\mathbf{x}(t + dt))$, i.e., $\mathbf{z}' = \arg \min_{\mathbf{z}'''} \mathcal{L}_{\mathbf{x}(t+dt)}(\mathbf{z}''')$. Let \mathbf{z}'' be the time-indexed schedule which first has \hat{z}_{ij} for sdt units of time and then exactly follow \mathbf{z}' on each machine i . Clearly, we have

$$f(\mathbf{x}(t)) - f(\mathbf{x}(t + dt)) \leq \mathcal{L}_{\mathbf{x}(t)}(\mathbf{z}'') - f(\mathbf{x}(t + dt))$$

Let's first consider the change of $(**)$ in the LP objective. Since GD processes jobs according to the distribution $\{\hat{z}_{ij}\}_j$ during $[t, t + dt)$, $(**)$ decreases by

$$\sum_{i,j} w_j \hat{z}_{ij} sdt = \tilde{W}_{(1)}^A sdt,$$

as we observed in the proof of Lemma 5.17.

We now turn our attention to the change of $(*)$, which is the fractional weighted completion time of jobs. By scheduling \hat{z} with speed s , every terms coefficient changes from $w_j/p_{ij}\tilde{t}$ to $w_j/p_{ij}(\tilde{t} - sdt)$. Thus, the decrease is at most the total fractional weight of all jobs times sdt , which is

$$(\tilde{W}^A(t) + (1/\epsilon - 1)\tilde{W}^O(t))sdt$$

By adding up the upper bounds on the change of $(*)$ and $(**)$ and using the fact that $\tilde{W}_{(1)}^A(t) \leq \tilde{W}^A(t)$, we obtain the lemma. \square

We now consider the change due to the adversary's processing. Note that the RHS consists of $\tilde{W}^A(t) := \sum_{j \in A(t)} w_j \frac{p_j^A(t)}{p_j}$ and $W^O(t) := \sum_{j \in O(t)} w_j$, which denote the total fractional remaining weight of jobs in $A(t)$ and the total integral weight of jobs in $O(t)$, respectively.

Lemma 5.19. $-\frac{d}{dt} f(\mathbf{p}^A(t) \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O(t)) \Big|_{\text{opt}} \leq (1/\epsilon - 1)\tilde{W}^A(t) + (1/\epsilon)(1/\epsilon - 1)W^O(t)$.

Proof. The proof of this lemma is very similar to that of Lemma 5.18. We keep the same notation for \mathbf{x} and \mathbf{z}' . Since the adversary processing job j with 1 speed is equivalent to processing all $(1/\epsilon - 1)$ copies of the same job at the same rate, we will pretend that all the copies are distinct jobs and the adversary has $(1/\epsilon - 1)$ -speed. Clearly, this only gives more power to the adversary. For simplicity we assume that the adversary schedules an integral matching during $[t, t + dt)$, since the extension of the analysis to fractional matching is straightforward. That is, the adversary processes exactly one job $j(i)$ on machine i during $[t, t + dt)$ and processes it on no other machines. Let \hat{z}^* denote this schedule. So, $z_{i,j(i)}^* = 1$ but $z_{i,j'}^* = 0$ for all $j' \neq j(i)$. Let \mathbf{z}'' be a schedule that has \hat{z}^* for $(1/\epsilon - 1)dt$ time steps, followed by \mathbf{z}' .

We note that $(**)$ decreases by

$$\sum_{i,j} w_j \hat{z}_{ij}^* (1/\epsilon - 1) dt = \sum_i w_{j(i)} (1/\epsilon - 1) dt \leq (1/\epsilon - 1) W^O(t) dt,$$

where the last inequality follows from the fact that $j(i) \neq j(i')$ for all $i \neq i'$.

By observing that the total fractional weight in $\mathbf{p}^A(t) \parallel (\frac{1}{\epsilon} - 1)\mathbf{p}^O(t)$ is $\tilde{W}^A(t) + (1/\epsilon - 1)\tilde{W}^O(t)$, we can upper bound the change of $(*)$ by

$$(1/\epsilon - 1)dt(\tilde{W}^A(t) + (1/\epsilon - 1)\tilde{W}^O(t))$$

We obtain the lemma by adding the two upper bounds and using the fact that $\tilde{W}^O(t) \leq W^O(t)$. \square

Corollary 5.20. Consider any time step t that no arrival or completion of jobs occurs. For $\epsilon < \frac{1}{8}$ and A is given $s = 1 + 2\epsilon$ -speed, we have $\frac{d}{dt}\Phi(t) \leq -\tilde{W}^A(t) + O(1/\epsilon^2)\tilde{W}^O(t)$.

Proof. From Lemma 5.17, Lemma 5.18, and Lemma 5.19, we have

$$\begin{aligned} \frac{d}{dt}\Phi(t) &\leq \frac{2}{\epsilon} \left(-s\tilde{W}^A(t) + \epsilon \left((2s + \frac{1}{\epsilon} - 1)\tilde{W}^A(t) + (s + \frac{1}{\epsilon})(\frac{1}{\epsilon} - 1)W^O(t) \right) \right) \\ &= \frac{2}{\epsilon} \left((-\epsilon + 4\epsilon^2)\tilde{W}^A(t) + O(1/\epsilon)W^O(t) \right) \\ &\leq -\tilde{W}^A(t) + O(1/\epsilon^2)W^O(t) \end{aligned}$$

\square

By Lemma 5.16 and $\Phi(T) = \Phi(0) = 0$, we have $\int_{t \geq 0} \frac{d}{dt}\Phi(t) \geq 0$. By Lemma 5.20, we have $\int_{t \geq 0} \tilde{W}^A(t) \leq O(1/\epsilon^2) \int_{t \geq 0} W^O(t)$. LHS is the total fractional weighted flow time of the algorithm and RHS is the total integral weighted flow time of the optimum schedule. Therefore, by Lemma 2.1, we obtain a $(1 + \epsilon)$ -speed, $O(1/\epsilon^3)$ -competitive algorithm for the integral objective.

5.7 Unrelated Machines via Immediate Dispatch In this section, we reproduce the result in [AGK12], which gave a $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive algorithm for minimizing total weighted flow time. The analysis of [AGK12] was based on dual fitting unlike [CGKM09], which gave a $(1 + \epsilon)$ -speed $O(1/\epsilon^2)$ -competitive algorithm using potential functions. We show that potential functions can also achieve an $O(1/\epsilon)$ -competitive ratio. At the high level, [AGK12] considered integral residual optimum unlike [CGKM09] that used fractional residual optimum.

Since we will use exactly the same algorithm as [AGK12] and obtain the same competitive ratio of $O(1/\epsilon)$, we will only sketch the analysis, focusing on how our approach leverages supermodularity and gradient descent explicitly.

The algorithm we consider is non-migratory and immediate-dispatch. Let $A_i(t)$ be the jobs assigned to machine i . Note that $\{A_i(t)\}_i$ will be a partition of $A(t)$. Let $\mathbf{p}^{A_i(t)}$ be the (vector of the) remaining sizes of jobs assigned to machine i at time t in our algorithm and define $\mathbf{p}^{O_i(t)}$ analogously for the optimum schedule.

We define $f(\mathbf{p}^{A_i(t)})$ as the residual optimum at time t of the jobs assigned to machine i . Formally, letting $p_{ij}(t) = p_j(t)/\lambda_{ij}$, $f(\mathbf{p}^{A_i(t)}) = \sum_j w_j \sum_{j' \leq j \text{ s.t. } w_{j'}/p_{ij'}(t) \geq w_j/p_{ij}(t)} p_{ij'}(t)$ is the minimum total weighted completion time of jobs in $A_i(t)$ pretending that each job $j \in A_i(t)$ has a remaining size $p_j(t)$, weight w_j , arrival

time 0; ties are broken in an arbitrary but fixed order. Since f is concerned with the single machine scheduling, it is straightforward to show its supermodularity. Then we let $\bar{f}(\mathbf{p}^A(t)) := \sum_i f(\mathbf{p}^{A_i}(t))$.

We can restate [AGK12] algorithm as follows: When a job j arrives at time t , we permanently assign the job j to the machine i that incurs the minimum increment of the residual optimum $\bar{f}(\mathbf{p}^A(t^-))$. Then, we process the jobs on each machine using gradient descent, i.e. follow the residual optimum schedule on the machine.

For the analysis, we use the following potential function, where $\bar{f}(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t))$ is defined analogously for jobs in A and O together.

$$\Phi(t) = \frac{2}{\epsilon} \left(\bar{f}(\mathbf{p}^A(t)) - \frac{1}{2} \bar{f}(\mathbf{p}^A(t) \parallel \mathbf{p}^O(t)) \right)$$

While we only consider the non-migratory optimum for brevity, it can be extended to the migratory case, following the approach described in [CGKM09].

Continuous Changes. If we focus on machine i , we only need to consider the change of $\frac{2}{\epsilon}(f(\mathbf{p}^{A_i}(t)) - \frac{1}{2}f(\mathbf{p}^{A_i}(t) \parallel \mathbf{p}^{O_i}(t)))$. This is exactly the same as the single machine scheduling where we can pretend that we only have jobs in $A_i(t)$ and the adversary has jobs in $O_i(t)$. Thus, using Corollary 4.6, we have that $\frac{d}{dt}\Phi(t)$ restricted to machine i is at most $-\sum_{j \in A_i(t)} w_j + (1 + \frac{2}{\epsilon}) \sum_{j \in O_i(t)} w_j$. Summing over all machines we have the following lemma.

Lemma 5.21. *Consider any time t when no job arrives or is completed by our algorithm or the adversary. When GD is given $1 + \epsilon$ -speed, $\frac{d}{dt}\Phi(t) \leq -W^A(t) + (1 + \frac{2}{\epsilon})W^O(t)$.*

Discrete Changes. The following lemma bounds the potential change due to a job's arrival or completion. Unlike the proofs in the previous section, we take a closer look at the machine to which the job is assigned.

Lemma 5.22. *If $\Phi(t)$ changes discontinuously due to a job's arrival or completion, the change is non-positive.*

Proof. For the same reason we used in the proof of Lemma 4.4, we only need to focus on the arrival case. Suppose a new job j arrives at time t . We use the same notations in Lemma 4.4. So j^A and j^O are job j 's copy in the schedule of A and O respectively. We assume wlog that job j^A is assigned to machine 1 by the algorithm. If j^O is assigned to the same machine 1, the change of $\bar{f}(\mathbf{p}^A \parallel \mathbf{p}^O)$ is at least

$$f(\mathbf{p}^{A_1} \parallel p_{1j}1_{j^A} \parallel \mathbf{p}^{O_1} \parallel p_{1j}1_{j^O}) - f(\mathbf{p}^{A_1} \parallel \mathbf{p}^{O_1}) \geq 2(f(\mathbf{p}^{A_1} \parallel p_{1j}1_{j^A}) - f(\mathbf{p}^{A_1})) \quad (5.21)$$

where the inequality holds due to the supermodularity of f .

Suppose j^O is assigned to another machine, say machine 2. Since the change only occurs on machines 1 and 2, the increase of $\bar{f}(\mathbf{p}^A \parallel \mathbf{p}^O)$ is at least

$$\begin{aligned} & f(\mathbf{p}^{A_1} \parallel p_{1j}1_{j^A} \parallel \mathbf{p}^{O_1}) - f(\mathbf{p}^{A_1} \parallel \mathbf{p}^{O_1}) + f(\mathbf{p}^{A_2} \parallel \mathbf{p}^{O_2} \parallel p_{2j}1_{j^O}) - f(\mathbf{p}^{A_2} \parallel \mathbf{p}^{O_2}) \\ & \geq f(\mathbf{p}^{A_1} \parallel p_{1j}1_{j^A}) - f(\mathbf{p}^{A_1}) + f(\mathbf{p}^{A_2} \parallel p_{2j}1_{j^O}) - f(\mathbf{p}^{A_2}), \end{aligned} \quad (5.22)$$

where the inequality follows from f 's supermodularity.

Recall that the algorithm assigns job j to the machine i that gives the minimum increment of f . Thus we have

$$f(\mathbf{p}^{A_1} \parallel p_{1j}1_{j^A}) - f(\mathbf{p}^{A_1}) \leq f(\mathbf{p}^{A_2} \parallel p_{2j}1_{j^O}) - f(\mathbf{p}^{A_2})$$

Then, Eqn.(5.22) is at least $2(f(\mathbf{p}_1^A \parallel p_{1j}1_{j^A}) - f(\mathbf{p}_1^A))$. Thus, in both cases, we have shown that $\bar{f}(\mathbf{p}^A \parallel \mathbf{p}^O)$ increases by at least $2(f(\mathbf{p}_1^A \parallel p_{1j}1_{j^A}) - f(\mathbf{p}_1^A))$. Since the change of $\bar{f}(\mathbf{p}^A)$ is exactly

$$f(\mathbf{p}^{A_1} \parallel p_{1j}1_{j^A}) - f(\mathbf{p}^{A_1}),$$

we can conclude that $\Phi(t)$'s change due to job j 's arrival is non-positive. \square

We combine Lemmas 5.21 and 5.22 as we combined Lemma 4.4 and Corollary 4.6 at the end of Section 4.1. As a result, we can show that the online algorithm is $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive for the total weighted flow time objective.

6 Other Related Work

Gross substitutes valuations were introduced by Kelso and Crawford in economics [KC82]. Interestingly, the same valuations were introduced in other areas under different names, such as M^{\natural} -concave functions [MS99], matroidal maps [DT95b, DT95a], and valuated matroids [DW92]. Thus, there are several equivalent ways to characterize them. The interested reader is referred to the nice survey [Lem17] and tutorial talk given in ACM EC 2018 [LTCom], and the extensive survey [Mur22].

We discuss two assumptions we make in this paper, clairvoyance and free preemption. Clairvoyant algorithms are assumed to know a job's size upon its arrival—in contrast, non-clairvoyant algorithms do not until completing the job. See [MPT94, KP00, ECBD03, PST04, EP12, IKMP14, IKM18, GGKS19] for some examples of non-clairvoyant algorithms. There have recently been work to design online algorithms when we are given jobs sizes that are not completely accurate [PSK18, IKMQP21, ALT21, ALT22, LM22], where the goal is to obtain a schedule almost as good as what the best clairvoyant algorithm can offer when the (predicted) sizes are not far from the actual sizes. In this paper we allow online algorithms to preempt jobs for free. In fact, the flow time minimization becomes computationally very hard without preemption. Thus, non-preemptive scheduling has been studied with resource augmentation even in the offline setting [BCK⁺07, CGKN04, ILMT15].

In this work we assume there are no precedence constraints among jobs. For online precedence constrained scheduling for total (weighted) flow time, see [ALLM16, GGKS19].

There exists a large literature on the “delay” variants of online problems, where the online algorithm can procrastinate its decision incurring delay cost, e.g., [AGGP17, ACKT18, AT20]. The main difference is that in our problem, we have capacitated resources that we can use to process jobs at each time, while the online algorithms pay extra cost for taking an action such as buying a set in their setting.

[DHL07] study Walrasian equilibrium for a market where jobs act as agents bidding for CPU times. Their work considers the single-machine setting, where jobs have different valuations depending on their own completion time. The focus is on the existence and computability of the equilibrium.

A Missing Proofs for Valuation Functions

A.1 Proof of Lemma 2.14

Proof. We have $v(\mathbf{x}) = \min_{\mathbf{q} \geq 0} \{\pi(\mathbf{q}) + \mathbf{q} \cdot \mathbf{x}\}$ (See [MS09] Theorem 26, the concavity implies this). Let $f(\mathbf{x}, \mathbf{q}) = \pi(\mathbf{q}) + \mathbf{q} \cdot \mathbf{x}$. We can show $\mathbf{q} \cdot \mathbf{x}$ is supermodular in \mathbf{q} and \mathbf{x} when $\mathbf{q}, \mathbf{x} \geq 0$ using on each coordinate the fact that $ab + a'b' \leq \max\{a, a'\} \max\{b, b'\} + \min\{a, a'\} \min\{b, b'\}$ for any scalars $a, a', b, b' \geq 0$. Then, we have $f(-\mathbf{x}, \mathbf{q}) = \pi(\mathbf{q}) - \mathbf{q} \cdot \mathbf{x}$, which is submodular in \mathbf{x} and \mathbf{q} . By taking the min over all \mathbf{q} , we know $v(-\mathbf{x})$ is submodular for the following reason. Let $g(\mathbf{x}, \mathbf{q}) := f(-\mathbf{x}, \mathbf{q})$. Let \mathbf{q} and \mathbf{q}' be such that $v(-\mathbf{x}) = g(\mathbf{x}, \mathbf{q})$ and $v(-\mathbf{x}') = g(\mathbf{x}', \mathbf{q}')$. We then have,

$$\begin{aligned} v(-\mathbf{x}) + v(-\mathbf{x}') &= g(\mathbf{x}, \mathbf{q}) + g(\mathbf{x}', \mathbf{q}') \\ &\geq g(\mathbf{x} \vee \mathbf{x}', \mathbf{q} \vee \mathbf{q}') + g(\mathbf{x} \wedge \mathbf{x}', \mathbf{q} \wedge \mathbf{q}') \quad [g(\mathbf{x}, \mathbf{q}) \text{ is submodular in both } \mathbf{x}, \mathbf{q}] \\ &\geq v(-(\mathbf{x} \vee \mathbf{x}')) + v(-(\mathbf{x} \wedge \mathbf{x}')), \end{aligned}$$

which proves $v(-\mathbf{x})$'s submodularity in \mathbf{x} . So, we have

$$\begin{aligned} v(\mathbf{x}) + v(\mathbf{y}) &= v(-(-\mathbf{x})) + v((- \mathbf{y})) \\ &\geq v(-(-\mathbf{x} \vee -\mathbf{y})) + v(-(-\mathbf{x} \wedge -\mathbf{y})) \quad [v(-\mathbf{x}) \text{ is submodular in } \mathbf{x}] \\ &= v(\mathbf{x} \wedge \mathbf{y}) + v(\mathbf{x} \vee \mathbf{y}) \end{aligned}$$

□

A.2 Concave Closure of GS: Proof of Theorem 2.15 Consider a fixed gross-substitute valuation $v : \{0, 1\}^n \rightarrow \mathbb{R}$.²⁵ We first observe that for any rational number $\delta > 0$, we can approximate v by a function $v' : \{0, 1\}^n \rightarrow \mathbb{Q}$ such that $|v(S) - v'(S)| \leq \delta$ for any $S \subseteq [n]$. This is because due to [LTCom] that shows that v is GS if and only if it satisfies a finite number of linear inequalities, we can set up a linear programming with

²⁵We interchangeably use $v(S)$ and $v(\mathbf{1}_S)$

variables $\{v'(S)\}_{S \subseteq [n]}$ satisfying the inequalities together with the approximation requirement for all S . The LP has a feasible solution, and particularly a rational number solution since $\delta \in \mathbb{Q}$. Thus, we assume $v : \{0, 1\}^n \rightarrow \mathbb{Q}$ henceforth.

We show that for any $\mathbf{x} \in \mathbb{Q}^n \cap [0, 1]^n$, we can view $v^+(\mathbf{x})$ as a convolution of multiples copies of v . The proof is not difficult, but we include it for completeness.

Lemma A.1. Define $\tilde{v}^+ : \mathbb{Q}^n \cap [0, 1]^n \rightarrow \mathbb{Q}$ such that

$$\tilde{v}^+(\mathbf{x}) = \sup_{\epsilon=1/k! : k \in \mathbb{Z}_{\geq 0}} \epsilon \max_{\mathbf{x}/\epsilon = \mathbf{x}_1 + \dots + \mathbf{x}_{1/\epsilon}} \sum_{i=1}^{1/\epsilon} v(\mathbf{x}_i)$$

where $\mathbf{x}_1, \dots, \mathbf{x}_{1/\epsilon} \in \{0, 1\}^n$. Then, for any $\mathbf{x} \in \mathbb{Q}^n \cap [0, 1]^n$,

$$\tilde{v}^+(\mathbf{x}) := v^+(\mathbf{x})$$

where $v^+(\mathbf{x}) = \max \{\sum_S v(S) \lambda_S : \sum_S \lambda_S \mathbf{1}_S = \mathbf{x}, \sum_S \lambda_S = 1, \lambda_S \geq 0\}$ is the concave closure of v .

Proof. Fix a $\mathbf{x} \in \mathbb{Q}^n \cap [0, 1]^n$. Let $\{\lambda_S\}_S$ be an optimal solution of $v^+(\mathbf{x})$; note that $\lambda_S \in \mathbb{Q}$ for all S . Consider an ϵ such that λ_S is an integer multiple of ϵ for all S . To show $\tilde{v}^+(\mathbf{x}) \geq v^+(\mathbf{x})$, we assign $\mathbf{1}_S$ to λ_S/ϵ copies of v . Then, $\tilde{v}^+(\mathbf{x}) \geq \epsilon \sum_S (\lambda_S/\epsilon) v(\mathbf{1}_S) = v^+(\mathbf{x})$. Conversely, fix an $\epsilon > 0$ such that $1/\epsilon$ is an integer, together with any $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{1/\epsilon}$ such that $\mathbf{x}/\epsilon = \sum_{i=1}^{1/\epsilon} \mathbf{x}_i$. Then, by setting $\lambda_S = \epsilon |\{\mathbf{x}_i = \mathbf{1}_S\}_i|$, one can immediately show $v^+(\mathbf{x}) \geq \tilde{v}^+(\mathbf{x})$. Finally, it is easy to see that $\tilde{v}(\mathbf{x})$ is well defined as in the definition of $\tilde{v}^+(\mathbf{x})$, the quantity monotonically increases as $k \rightarrow \infty$. \square

Remark A.2. Strictly speaking, when we create $1/\epsilon$ copies of v , we also create $1/\epsilon$ copies of each item. Each copy of v is assumed to have no benefit from having more than one copy of any item. It is easy to check that this valuation function is GS over all the item copies.

The following corollary is immediate from the proof of the above lemma.

Corollary A.3. For any $\mathbf{x} \in \mathbb{Q}^n \cap [0, 1]^n$, there exists $\epsilon > 0$ such that $v^{+\epsilon}(\mathbf{x}) = \tilde{v}^+(\mathbf{x}) = v^+(\mathbf{x})$, where

$$v^{+\epsilon}(\mathbf{x}) = \epsilon \max_{\mathbf{x}/\epsilon = \mathbf{x}_1 + \dots + \mathbf{x}_{1/\epsilon}} \sum_{i=1}^{1/\epsilon} v(\mathbf{x}_i)$$

From Lemma 2.13, it is sufficient to show the dual profit of v^+ is submodular.

Lemma A.4. The dual profit $\pi^+(\mathbf{q}) = \max_{\mathbf{x} \in \mathbb{R}^n} v^+(\mathbf{x}) - \mathbf{q} \cdot \mathbf{x}$ is submodular.

Proof. We first assume $\mathbf{q} \in \mathbb{Q}^n$. When $\mathbf{q} \in \mathbb{Q}^n$, we know $\mathbf{x} = \arg \max_{\mathbf{x}' \in \mathbb{R}^n} v^+(\mathbf{x}') - \mathbf{q} \cdot \mathbf{x}'$ is in \mathbb{Q}^n . Thus, from the previous lemma and corollary, we have $v^+(\mathbf{x}) = \tilde{v}^+(\mathbf{x})$. Further, for some $\epsilon > 0$, we have $v^{+\epsilon}(\mathbf{x}) = v^+(\mathbf{x})$. Then, we have,

$$\begin{aligned} \pi^+(\mathbf{q}) &= \max_{\mathbf{x} \in \mathbb{Q}_+^n} v^{+\epsilon}(\mathbf{x}) - \mathbf{q} \cdot \mathbf{x} \\ &= \max_{\mathbf{x} \in \mathbb{Q}_+^n} \epsilon \max_{\mathbf{x}/\epsilon = \mathbf{x}_1 + \dots + \mathbf{x}_{1/\epsilon}} \sum_{i=1}^{1/\epsilon} v(\mathbf{x}_i) - \mathbf{q} \cdot \mathbf{x}_i \\ &= \epsilon \max_{\mathbf{x}_1, \dots, \mathbf{x}_{1/\epsilon} \in \{0, 1\}^n} \sum_{i=1}^{1/\epsilon} v(\mathbf{x}_i) - \mathbf{q} \cdot \mathbf{x}_i \\ &= \epsilon(1/\epsilon)\pi(\mathbf{q}) \\ &= \pi(\mathbf{q}) \end{aligned}$$

Thus, we have shown that π^+ coincides with π on all rational vectors \mathbf{q} . Due to continuity, it does on all real vectors \mathbf{q} . Since v is a gross-substitute valuation, π is submodular. Therefore, π^+ is submodular, as desired. \square

A.3 Generalized Flow is Linear Substitute To show that v is LS, from Lemma 2.13, we know that it is equivalent to showing its dual profit $\pi(\mathbf{q})$ is submodular w.r.t price vector $\mathbf{q} \in \mathbb{R}^n$. The following lemma is due to Fleischer [Fle10] based on LP primal-dual. For a combinatorial proof, see [LK11].

Lemma A.5 (Fleischer [Fle10]). *Let $g(S)$ denote the (cost of the) min-cost flow with non-negative excess at the set of nodes $S \subseteq V \setminus V^-$, that is the min-cost flow with excess vector b' defined as $b'_v = b_v$ for $v \in S \cup V^-$; and $b'_v = 0$ for the other nodes. Then, g is a supermodular.*

Intuitively, g measures the cost of the min-cost flow of the underlying network G when we ‘switch’ on a subset S of sources V^- ; all the other sources do not generate any flow. As we add more sources, they create more congestion, which causes supermodularity.

We show that Lemma A.5 implies the valuation function we consider is LS. The proof idea is relatively simple. Due to Lemma 2.13, it suffices to show the dual profit function is submodular. Towards that end, we simulate increasing the price of a job’s processing unit by adding/activating new source nodes associated with the job.

We first show submodularity when the prices are rational numbers, i.e., $\mathbf{q} \in \epsilon\mathbb{Z}^n$, for any $\epsilon > 0$ such that $1/\epsilon$ is an integer, and extend it to arbitrary prices.

Lemma A.6. *For any $\epsilon \in 1/\mathbb{Z}_+$, $\pi^{+\epsilon}(\mathbf{q}) = \max_{\mathbf{x} \in \mathbb{R}_+^n} v(\mathbf{x}) - \mathbf{q} \cdot \mathbf{x}$ is submodular where $\mathbf{q} \in \epsilon\mathbb{Z}_+^n$.*

Proof. We construct an auxiliary network \tilde{G} such that the min-cost flow of \tilde{G} captures the dual profit $-\pi^\epsilon$ as follows. We extend G to \tilde{G} by adding nodes and arcs as follows. Without declaration, the flow gain γ_e is 1 and the capacity u_e is ∞ for all new arcs e , and the excess b_v are ∞ for all new nodes. Consider the source node for job j in G . Let q_j be its price. First, add a new node j_0 to G and connect it to j by an arc (j_0, j) with cost $-c_j$. Then we add a series of arcs: for every $j \in \mathbb{Z}_+$, we add a new node j_i , and connect it to j_{i-1} with cost ϵ . Finally, we set the cost of each arc in G to be 0.

Then we let $g(S)$ denote the min-cost flow with nonnegative excess at the set of nodes S , that is the excess b' defined as $b'_v = b_v$ for all $v \in S \cup V^-$; and $b'_v = 0$ for the other nodes. Now fix any price vector $\mathbf{q} \in \epsilon\mathbb{Z}_+^n$. Let $S_{\mathbf{q}}$ be a set of nodes that contains $\{j_{q_j/\epsilon}, j_{q_j/\epsilon+1}, \dots\}$ for each $j \in J$. Then we notice that $-\pi^{+\epsilon}(\mathbf{q})$ is equal to $g(S_{\mathbf{q}})$. Consider two price vectors $\mathbf{q}, \mathbf{q}' \in \epsilon\mathbb{Z}_+^n$ and $\mathbf{q} \leq \mathbf{q}'$, and a job j . Note that $\pi^{+\epsilon}(\mathbf{q} + \epsilon \mathbf{1}_j) - \pi^{+\epsilon}(\mathbf{q}) = -g(S_{\mathbf{q}+\epsilon \mathbf{1}_j}) + g(S_{\mathbf{q}})$ and $S_{\mathbf{q}+\epsilon \mathbf{1}_j} + \{j_{q_j/\epsilon}\} = S_{\mathbf{q}}$. Because $\mathbf{q} \leq \mathbf{q}'$, we have $S_{\mathbf{q}'} \subseteq S_{\mathbf{q}}$. Then the lemma follows from the supermodularity of g due to Lemma A.5. \square

Lemma A.7. *The dual profit $\pi : \mathbb{R}_+^n \rightarrow \mathbb{R}$ is submodular.*

Proof. For any feasible \mathbf{x} of v , we can write $\pi(\mathbf{q}) = \max_{\mathbf{x}} (\mathbf{c} - \mathbf{q}) \cdot \mathbf{x}$. Then, it is clear that π is continuous.

Suppose that π is not submodular. Let $\mathbf{q}, \mathbf{b} \in \mathbb{R}_+^n$ be the price vectors that violate the submodularity. Then, we have

$$\pi(\mathbf{q}) + \pi(\mathbf{b}) < \pi(\mathbf{q} \vee \mathbf{b}) + \pi(\mathbf{q} \wedge \mathbf{b})$$

Let Δ be the difference of the above inequality. From the continuity of π , for some infinitesimally small $\epsilon' > 0$, we have $|\pi(\mathbf{q}) - \pi(\mathbf{q}')|, |\pi(\mathbf{b}) - \pi(\mathbf{b}')|, |\pi(\mathbf{q} \vee \mathbf{b}) - \pi(\mathbf{q}' \vee \mathbf{b}')|$ and $|\pi(\mathbf{q} \wedge \mathbf{b}) - \pi(\mathbf{q}' \wedge \mathbf{b}')|$ are less than ϵ' , for some δ such that $\|\mathbf{q} - \mathbf{q}'\|_1 < \delta$ and $\|\mathbf{b} - \mathbf{b}'\|_1 < \delta$ where $\mathbf{q}', \mathbf{b}' \in \epsilon\mathbb{Z}_+^n$ for some $\epsilon \in 1/\mathbb{Z}_+$. Since $\Delta - 4\epsilon' > 0$, we have

$$\pi^{+\epsilon}(\mathbf{q}') + \pi^{+\epsilon}(\mathbf{b}') < \pi^{+\epsilon}(\mathbf{q}' \vee \mathbf{b}') + \pi^{+\epsilon}(\mathbf{q}' \wedge \mathbf{b}')$$

This would contradict the submodularity of $\pi^{+\epsilon}$ from Lemma A.6, and thus we have the lemma. \square

B Gradient Descent is Not a Panacea

In this section, we briefly show that gradient descent is not $O(1)$ -competitive with any $O(1)$ -speed. In the (pull-based) broadcast scheduling [AF98, KPV00, BCKN05, BKN10, IM12], the server stores some n pages of useful information. At each time some jobs arrive, each asking for a page. For simplicity, assume all pages are unit-sized. When the server broadcasts a page, it satisfies all jobs requesting the same page and it can broadcast only one page at a time. Scalable algorithms are known for this problem [BKN10, IM12].

Suppose gradient descent has s -speed and s is an integer. Consider the following instance. There are special pages, e_1, e_2, \dots, e_s . At each time, a total of $2s$ jobs arrive, two jobs asking for each special page. Further, in every $s + 1$ time steps, one job arrives asking for a distinct page. Suppose the instance ends at time $(s + 1)T$.

At each time, gradient descent processes the $2s$ jobs requesting the special pages, having no time to work on any other jobs. Thus, at time T , it has $\Omega(T)$ jobs alive, which have $\Omega(T)$ flow time on average. Therefore, gradient descent's total flow time is $\Omega(T^2)$. On the other hand, the adversary can repeat the following: Broadcast each of the special s pages and then the unique other page requested. It is easy to see that it satisfies every job within $O(1)$ -time step. Therefore, its total flow time is $O(T)$.

C Making Gradient Descent Run in Polynomial Time

The high-level idea is to replace time slots with intervals of exponentially increasing lengths to reduce the number of variables to consider. This idea is commonly used in approximation algorithms to make a time-indexed LP compact. While it is straightforward to see it only losing $1 + \epsilon$ factor in the approximation ratio, here we should be more careful as we need to show the change of the approximate LP optimum (residual optimum) is $1 + \epsilon$ -approximate as opposed to what we obtain with the exact residual optimum.

We define time intervals $\mathcal{I} := \{I_h := [a_h, a_{h+1}) \mid h \geq 1\}$, where $a_1 < a_2 < \dots$ are the integers in $[\frac{10}{\rho^2}] \cup \{\lfloor \frac{10}{\rho^2}(1 + \rho)^l \rfloor \mid l \geq 1\}$, in increasing order, for some small constant $\rho > 0$. We will set ρ later and we will assume $1/\rho$ is an integer. Let $|I|$ denote the length of the interval I ; more precisely, the number of integer time steps in I . The following properties of \mathcal{I} are immediate from the construction.

Observation C.1. *The collection \mathcal{I} of intervals has the following properties. Let $I_{H'}$ be the first interval of length greater than 1.*

1. For all $h \geq 1$, $|I_h| \leq |I_{h+1}|$.
2. All intervals $I_1, I_2, \dots, I_{H'-1}$ have length 1.
3. For all $h \geq H'$, $|I_h| \geq \frac{1}{1+2\rho}|I_{h+1}|$.

In the residual optimum, we will pretend that time steps in each interval in \mathcal{I} are all identical. Towards this end, define $g(t)$ to be a_h where $I_h := [a_h, a_{h+1})$ is the unique interval in \mathcal{I} including t . So, we will consider the following LP.

$$\begin{aligned} \text{Approximate Residual (Time-Indexed) LP } \mathcal{L}_{\mathbf{x}}^{\mathcal{P}}: \quad f(\mathbf{x}) := \min \quad & \sum_{j \in A(t)} \frac{w_j}{p_j} \sum_{\tilde{t} \geq 1} g(\tilde{t}) \cdot z_j(\tilde{t}) \\ & \sum_{\tilde{t} \geq 1} z_j(\tilde{t}) d\tilde{t} = x_j \quad \forall j \in A(t) \\ & \mathbf{z}(\tilde{t}) \in \mathcal{P} \quad \forall \tilde{t} \geq 1 \end{aligned} \quad (\text{C.1})$$

Note that in reality, $g(\tilde{t})$ is the same for all $\tilde{t} \in I_h$, so we can aggregate the variables $\{z_j(\tilde{t})\}_{\tilde{t} \in I_h}$ into one; this approximate LP is of compact size. However, we take this full expanded view for the sake of analysis, assuming wlog that $z_j(\tilde{t})$ is the same for all \tilde{t} in each I_h , for any fixed job j .

It is easy to see that replacing \tilde{t} with $g(\tilde{t})$ does not change the residual objective's supermodularity and monotonicity. The only properties we need to check is that the residual objective's change rate remains the same within $1 + \epsilon$ factor, under GD and the adversary, which can be easily offset by extra $1 + O(\epsilon)$ -speed augmentation. So, henceforth we aim to show it.

GD remains effective in decreasing the residual optimum. We first show that processing following the optimum solution to the above LP in the first time step decreases the objective by an amount that is almost equal to the total fractional weight of all jobs. Let $\mathbf{z} := \langle \mathbf{z}_1, \mathbf{z}_2, \dots \rangle$ be the optimum solution to $\mathcal{L}_{\mathbf{x}}$. We may use inequalities in the subscript to consider a partial schedule of \mathbf{z} ; for example, $\mathbf{z}_{\geq 2} := \langle \mathbf{z}_2, \mathbf{z}_3, \dots \rangle$.

Lemma C.2. *Let $\mathbf{z} := \langle \mathbf{z}_1, \mathbf{z}_2, \dots \rangle$ be the optimum solution to $\mathcal{L}_{\mathbf{x}}$. Then, $f(\mathbf{x}) - f(\mathbf{x} - \mathbf{z}_1) \geq \mathcal{L}_{\mathbf{x}}(\mathbf{z}_{\geq 1}) - \mathcal{L}_{\mathbf{x}-\mathbf{z}_1}(\mathbf{z}_{\geq 2}) \geq \frac{1}{1+2\rho} \tilde{W}(t)$.*

Before beginning proving the lemma, we first make a simple observation, which will be useful throughout this section. Intuitively, one wants to complete more fractional weights earlier to minimize the objective, which is formally stated below.

Claim C.3. *Let $V(\tau) := \sum_j \frac{w_j}{p_j} z_j(\tau)$. We have $V(a_h) \geq V(a_{h+1})$.*

Proof. Otherwise, swapping $\mathbf{z}(a_h)$ and $\mathbf{z}(a_{h+1})$ decreases the LP objective, contradicting \mathbf{z} 's optimality. \square

Proof of Lemma C.2. The first inequality, $f(\mathbf{x}) - f(\mathbf{x} - \mathbf{z}_1) \geq \mathcal{L}_{\mathbf{x}}(\mathbf{z}_{\geq 1}) - \mathcal{L}_{\mathbf{x} - \mathbf{z}_1}(\mathbf{z}_{\geq 2})$, follows from the observation that $\mathbf{z}_{\geq 2}$ is a feasible schedule to $\mathcal{L}_{\mathbf{x} - \mathbf{z}_1}$. Let I_H be the last interval where $z_j(a_H)$ has a non-zero value for some j . We then have,

$$\begin{aligned}
& \mathcal{L}_{\mathbf{x}}(\mathbf{z}_{\geq 1}) - \mathcal{L}_{\mathbf{x} - \mathbf{z}_1}(\mathbf{z}_{\geq 2}) \\
&= \sum_j \frac{w_j}{p_j} \sum_{h \in [H]} (a_h - a_{h-1}) z_j(a_h) = \sum_j \frac{w_j}{p_j} \sum_{h \in [H]} |I_{h-1}| z_j(a_h) \\
&\geq \sum_j \frac{w_j}{p_j} \sum_{h \leq H'} |I_{h-1}| z_j(a_h) + \sum_j \frac{w_j}{p_j} \sum_{H' < h \leq H} |I_{h-1}| z_j(a_h) \\
&\geq \sum_j \sum_{h \leq H'} \frac{w_j}{p_j} z_j(a_h) + \sum_j \frac{w_j}{p_j} \sum_{H' < h < H} \frac{1}{1+2\rho} |I_h| z_j(a_h) \quad [\text{Observation C.1}] \\
&= \sum_{\tau < a_{H'}} V(\tau) + \frac{1}{|I_{H'}|} \sum_{a_{H'} \leq \tau < a_{H'+1}} V(\tau) + \frac{1}{1+2\rho} \sum_{\tau \geq a_{H'+1}} V(\tau) \\
&\geq \frac{1}{1+2\rho} \sum_{\tau} V(\tau) \\
&= \frac{1}{1+2\rho} \tilde{W}(t)
\end{aligned}$$

The penultimate inequality follows from Claim C.3 and $|I_{H'}| \leq 2\rho a_{H'}$. \square

By setting ρ small enough (and using slightly more speed augmentation), we will argue that GD effectively decreases the residual optimum.

No need to recompute the residual optimum until a new job arrives. In the above, we only showed the GD effectively decreases the residual objective for one time step. Here, we further argue that we can just process $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{\hat{t}}$ until a new job arrives at time $t + \hat{t}$. To this end, we *pretend* that the residual changes by

$$\mathcal{L}_{\mathbf{x} - \mathbf{z}_{\leq \tau-1}}(\mathbf{z}_{\geq \tau}) - \mathcal{L}_{\mathbf{x} - \mathbf{z}_{\leq \tau}}(\mathbf{z}_{\geq \tau+1}) \quad (\text{C.2})$$

when we process z_{τ} at time $t + \tau$. This is because until a new job arrives at time τ' , the residual decreases by at least $\mathcal{L}_{\mathbf{x} - \mathbf{z}_{\leq 0}}(\mathbf{z}_{\geq 1}) - \mathcal{L}_{\mathbf{x} - \mathbf{z}_{\leq \tau'}}(\mathbf{z}_{\geq \tau'+1})$, and we can decompose it as a telescopic sum, where we pretend that the residual decreases by the amount shown in Eqn. (C.2) at each time τ .

No algorithm can decrease the residual optimum much more than GD. We need to argue that no algorithm can decrease the residual optimum by more than $\tilde{W}(t)$ (with speed 1). To see this, fix an algorithm and suppose it processes \mathbf{z}_1 in the first time slot, and the optimum schedule for the remaining jobs sizes, $\mathbf{x} - \mathbf{z}_1$, is $\langle \mathbf{z}_2, \mathbf{z}_3, \dots \rangle$. Our goal is to upper bound,

$$\begin{aligned}
f(\mathbf{x}) - \mathcal{L}_{\mathbf{x} - \mathbf{z}_1}(\mathbf{z}_{\geq 2}) &\leq \mathcal{L}_{\mathbf{x} - \mathbf{z}_0}(\mathbf{z}_{\geq 1}) - \mathcal{L}_{\mathbf{x} - \mathbf{z}_1}(\mathbf{z}_{\geq 2}) \quad [\mathbf{z}_{\geq 1} \text{ is a feasible schedule to } \mathcal{L}_{\mathbf{x}} \text{ and } \mathbf{z}_0 = 0] \\
&\leq \sum_j \frac{w_j}{p_j} \sum_{h \in [H]} (a_h - a_{h-1}) z_j(a_h) = \sum_j \frac{w_j}{p_j} \sum_{h \in [H]} |I_{h-1}| z_j(a_h) \\
&\leq \sum_j \frac{w_j}{p_j} \sum_{h \in [H]} |I_h| z_j(a_h) \quad [\text{Observation C.1}] \\
&= \sum_{\tau} V(\tau) = \tilde{W}(t)
\end{aligned}$$

Lemma C.4. No algorithm can decrease the residual optimum by more than $\tilde{W}(t)$ at a time t .

Putting the Pieces Together. We use the same potential function we used in the proof of Theorem 3.3, except that f is replaced with the approximate function we consider in Eqn. (C.1). All the other properties such as supermodularity remain to be satisfied with the approximate residual LP. Thus, the former analysis for

discrete events such as jobs arrival or completion continues to hold true (Lemmas 4.7 and 4.8). The bound in Lemma 4.9 becomes $\frac{d}{dt}f(\mathbf{p}^A(t)) = -\frac{s}{1+2\rho}\tilde{W}^A(t)$ thanks to Lemma C.2. The bound in Lemma 4.10 remains unchanged due to Lemma C.4. Then, assuming the online algorithm is given $(1+\epsilon)(1+2\rho)$ -speed, the bound in Corollary 4.11 becomes $\frac{d}{dt}\Phi(t) \leq -\tilde{W}^A(t) + \frac{1}{\epsilon}((1+\epsilon)(1+2\rho)+2)\tilde{W}^O(t)$. We can set ρ to ϵ to obtain an algorithm that is $(1+\epsilon)(1+2\epsilon)$ -speed $O(1/\epsilon)$ -competitive for the fractional objective. By scaling ϵ appropriately, we have Theorem 3.3, even with the approximate residual optimum. Thus, we have shown polynomial time algorithms achieving the theorem—more precisely, assuming that we have a poly-time separation oracle for whether $\mathbf{z} \in \mathcal{P}$ or not. As a result, We also have polynomial time algorithms for Theorem 3.4 as the proof of the theorem reduces the resource view to the processing rate view.

References

- [ABF⁺17] S. Anand, Karl Bringmann, Tobias Friedrich, Naveen Garg, and Amit Kumar. Minimizing maximum (weighted) flow-time on related and unrelated machines. *Algorithmica*, 77(2):515–536, 2017.
- [AC04] Lawrence M Ausubel and Peter Cramton. Auctioning many divisible goods. *Journal of the European Economic Association*, 2(2-3):480–493, 2004.
- [ACKT18] Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay–clairvoyance is not required. *arXiv preprint arXiv:1807.08543*, 2018.
- [AF98] Demet Aksoy and Michael Franklin. Scheduling for large-scale on-demand data broadcasting. In *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98*, volume 2, pages 651–659. IEEE, 1998.
- [AGGP17] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 551–563, 2017.
- [AGK12] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.
- [ALLM16] Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 176–189. SIAM, 2016.
- [ALT21] Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1070–1080, 2021.
- [ALT22] Yossi Azar, Stefano Leonardi, and Noam Touitou. Distortion-oblivious algorithms for minimizing flow time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 252–274. SIAM, 2022.
- [AT20] Yossi Azar and Noam Touitou. Beyond tree embeddings—a deterministic framework for network design with deadlines or delay. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1368–1379. IEEE, 2020.
- [BCK⁺07] Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Clifford Stein, and Baruch Schieber. Non-preemptive min-sum scheduling with resource augmentation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 614–624. IEEE Computer Society, 2007.
- [BCKN05] Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph Seffi Naor. Approximating the average response time in broadcast scheduling. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 215–221. Society for Industrial and Applied Mathematics, 2005.
- [BCP13] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Trans. Algorithms*, 9(2):18:1–18:14, 2013.
- [BEY05] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.
- [BKN10] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. In *Automata, Languages and Programming: 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I 37*, pages 324–335. Springer, 2010.
- [BPS09] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.
- [BSN09] Niv Buchbinder and Joseph (Seffi) Naor. The design of competitive online algorithms via a primal: Dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2–3):93–263, February 2009.

- [CGKM09] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *STOC*, pages 679–684, 2009.
- [CGKN04] Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 81–90. IEEE Computer Society, 2004.
- [CPU] CPU partitioning. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/system_partitioning.
- [DGBL00] Xiaotie Deng, Nian Gu, Tim Brecht, and Kaicheng Lu. Preemptive scheduling of parallel jobs on multiprocessors. *SIAM Journal on Computing*, 30(1):145–160, 2000.
- [DH14] Nikhil R. Devanur and Zhiyi Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1123–1140, 2014.
- [DHL07] Xiaotie Deng, Li-Sha Huang, and Minming Li. On walrasian price of CPU time. *Algorithmica*, 48(2):159–172, 2007.
- [DT95a] Andreas WM Dress and Werner Terhalle. Rewarding maps: On greedy optimization of set functions. *Advances in Applied Mathematics*, 16(4):464–483, 1995.
- [DT95b] Andreas WM Dress and Werner Terhalle. Well-layered maps—a class of greedily optimizable set functions. *Applied Mathematics Letters*, 8(5):77–80, 1995.
- [DW92] Andreas WM Dress and Walter Wenzel. Valuated matroids. *Advances in Mathematics*, 93(2):214–250, 1992.
- [ECBD03] Jeff Edmonds, Donald D Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *Journal of Scheduling*, 6(3):231–250, 2003.
- [Edm99] Jeff Edmonds. Scheduling in the dark. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 179–188. ACM, 1999.
- [EP12] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. *ACM Transactions on Algorithms*, 8(3):28, 2012.
- [Fle10] Lisa Fleischer. Data center scheduling, generalized flows, and submodularity. In *Proceedings of the Seventh Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2010, Austin, Texas, USA, January 16, 2010*, pages 56–65. SIAM, 2010.
- [GGKS19] Naveen Garg, Anupam Gupta, Amit Kumar, and Sahil Singla. Non-clairvoyant precedence constrained scheduling. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [GKP12] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In Thomas Erlebach and Giuseppe Persiano, editors, *Approximation and Online Algorithms - 10th International Workshop, WAOA 2012, Ljubljana, Slovenia, September 13-14, 2012, Revised Selected Papers*, volume 7846 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2012.
- [GS99] Faruk Gul and Ennio Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87(1):95–124, 1999.
- [GZH⁺11] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation, NSDI’11*, pages 24–24, 2011.
- [HSSW97] Leslie A Hall, Andreas S Schulz, David B Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of operations research*, 22(3):513–544, 1997.
- [IKM18] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. *J. ACM*, 65(1):3:1–3:33, 2018.
- [IKMP14] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 531–540. IEEE, 2014.
- [IKMQP21] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 285–294, 2021.
- [ILMT15] Sungjin Im, Shi Li, Benjamin Moseley, and Eric Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines [extended abstract]. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1070–1086. SIAM, 2015.
- [IM11] Sungjin Im and Benjamin Moseley. An online scalable algorithm for minimizing ℓ_k -norms of weighted flow time on unrelated machines. In *SODA*, 2011.

- [IM12] Sungjin Im and Benjamin Moseley. An online scalable algorithm for average flow time in broadcast scheduling. *ACM Trans. Algorithms*, 8(4):39:1–39:17, 2012.
- [IM15] Sungjin Im and Benjamin Moseley. Scheduling in bandwidth constrained tree networks. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 171–180. ACM, 2015.
- [IMP11] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011.
- [IMPP19] Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Manish Purohit. Matroid coflow scheduling. In *ICALP*, 2019.
- [JS15] Brendan Jennings and Rolf Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23:567–619, 2015.
- [JV10] Kamal Jain and Vijay V. Vazirani. Eisenberg-gale markets: Algorithms and game-theoretic properties. *Games Econ. Behav.*, 70(1):84–106, 2010.
- [KC82] Alexander S. Kelso and Vincent P. Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica*, 50(6):1483–1504, 1982.
- [KMT98] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *The Journal of the Operational Research Society*, 49(3):pp. 237–252, 1998.
- [KP00] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.
- [KPV00] Bala Kalyanasundaram, Kirk Pruhs, and Mahendran Velauthapillai. Scheduling broadcasts in wireless networks. In *ESA*, pages 290–301. Springer, 2000.
- [LAHW97] David B. Shmoys Leslie A. Hall, Andreas S. Schulz and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.
- [Lem17] Renato Paes Leme. Gross substitutability: An algorithmic survey. *Games and Economic Behavior*, 106:294–316, 2017.
- [LK11] Jian Li and Samir Khuller. Generalized machine activation problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 80–94. SIAM, 2011.
- [LLN06] Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.
- [LM22] Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 357–368. ACM, 2022.
- [LST90] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46:259–271, 1990.
- [LTCom] Renato Paes Leme and Inbal Talgam-Cohen. Gross substitutes tutorial, 2018 (Slides can be found at the second author’s webpage: <http://www.inbaltalgam.com>).
- [MPT94] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994.
- [MS99] Kazuo Murota and Akiyoshi Shioura. M-convex function on generalized polymatroid. *Mathematics of operations research*, 24(1):95–105, 1999.
- [MS09] Paul Milgrom and Bruno Strulovici. Substitute goods, auctions, and equilibrium. *Journal of Economic theory*, 144(1):212–247, 2009.
- [Mur22] Kazuo Murota. Discrete convex analysis: A tool for economics and game theory. *arXiv preprint arXiv:2212.03598*, 2022.
- [Nas50] J. Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.
- [PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.
- [PST04] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. 2004.
- [PSTW02] Cynthia A Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.
- [Sch03] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Springer-Verlag, Berlin, 2003.
- [Sku01] Martin Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM (JACM)*, 48(2):206–242, 2001.
- [SS99] Jay Sethuraman and Mark S Squillante. Optimal scheduling of multiclass parallel machines. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 963–964, 1999.
- [SS02] Andreas S Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002.
- [ST15] Akiyoshi Shioura and Akihisa Tamura. Gross substitutes condition and discrete concavity for multi-unit valuations: a survey. *Journal of the Operations Research Society of Japan*, 58(1):61–103, 2015.