Improved Scheduling with a Shared Resource

Christoph Damerius(✉), Peter Kling, and Florian Schneider

University of Hamburg, 22527 Hamburg, Germany
{damerius,fschneider}@informatik.uni-hamburg.de,
peter.kling@uni-hamburg.de

Abstract. We consider the following shared-resource scheduling problem: Given a set of jobs J, for each $j \in J$ we must schedule a job-specific processing volume of $v_j > 0$. A total resource of 1 is available at any time. Jobs have a resource requirement $r_j \in [0, 1]$, and the resources assigned to them may vary over time. However, assigning them less will cause a proportional slowdown.

We consider two settings. In the first, we seek to minimize the makespan in an online setting: The resource assignment of a job must be fixed before the next job arrives. Here we give an optimal $e/(e-1)$-competitive algorithm with runtime $O(n \log n)$. In the second, we aim to minimize the total completion time. We use a continuous linear programming (CLP) formulation for the fractional total completion time and combine it with a previously known dominance property from malleable job scheduling to obtain a lower bound on the total completion time. We extract structural properties by considering a geometrical representation of a CLP's primal-dual pair. We combine the CLP schedule with a greedy schedule to obtain a $(3/2 + \varepsilon)$-approximation for this setting. This improves upon the so far best-known approximation factor of 2.

Keywords: Approximation Algorithm · Malleable Job Scheduling · Makespan · List Scheduling · Completion Time · Continuous Linear Program

1 Introduction

Efficient allocation of scarce resources is a versatile task lying at the core of many optimization problems. One of the most well-studied resource allocation problems is parallel processor scheduling, where a number of *jobs* need (typically at least temporarily exclusive) access to one or multiple *machines* to be completed. The problem variety is huge and might depend on additional constraints, parameters, available knowledge, or the optimization objective (see [14]).

In the context of computing systems, recent years demonstrated a bottleneck shift from *processing power* (number of machines) towards *data throughput*. Indeed, thanks to cloud services like AWS and Azure, machine power is available in

Full Version (Preprint): https://arxiv.org/abs/2310.05732.

W. Wu and J. Guo (Eds.): COCOA 2023, LNCS 14461, pp. 154–167, 2024.
https://doi.org/10.1007/978-3-031-49611-0_11

abundance while data-intensive tasks (e.g., training LLMs like ChatGPT) rely on a high data throughput. If the bandwidth of such data-intensive tasks is, say halved, they may experience a serious performance drop, while computation-heavy tasks care less about their assigned bandwidth. In contrast to the number of machines, throughput is (effectively) a *continuously* divisible resource whose distribution may be easily changed *at runtime*. This opens an opportunity for adaptive redistribution of the available resource as jobs come and go. Other examples of similarly flexible resources include power supply or the heat flow in cooling systems.

This work adapts formal models from a recent line of work on such flexible resources [1,5,13] and considers them under new objectives and settings. Classical *resource constrained scheduling* [8,11,15,16] assumes an "all-or-nothing" mentality (a job can be processed if it receives its required resource but is not further affected). One key aspect of the model we consider is the impact of the amount of received resource on the jobs' performance (sometimes referred to as *resource-dependent processing times* [9–12]). The second central aspect is that we allow a job's resource assignment to change while the job is running.

1.1 Model Description and Preliminaries

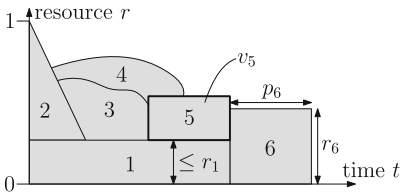
We consider a scheduling setting where a set $J = [n] := \{1, 2, \dots, n\}$ of $n \in \mathbb{N}$ *jobs* compete for a finite, shared resource in order to be processed. A *schedule* $R = (R_j)_{j \in J}$ consists of an (integrable) function $R_j \colon \mathbb{R}_{\geq 0} \to [0, 1]$ for each $j \in J$ (the job's *resource assignment*) that returns what fraction of the resource is assigned to j at time $t \in \mathbb{R}_{\geq 0}$. We use $R(t) = (R_j(t))_{j \in J}$ to refer to j's *resource distribution* at time t and $\bar{R}(t) := \sum_{j \in J} R_j(t)$ for the *total resource usage* at time t. Each $j \in J$ comes with a (*processing*) *volume* $v_j \in \mathbb{R}_{\geq 0}$ (the total amount of resource the job needs to receive over time in order to be completed) and a *resource requirement* $r_j \in [0, 1]$ (the maximum fraction of the resource the job can be assigned). We say a schedule $R = (R_j)_{j \in J}$ is *feasible* if:

- the resource is never overused: $\forall t \in \mathbb{R}_{\geq 0} \colon \bar{R}(t) \leq 1$,
- a job never receives more than its resource requirement: $\forall t \in \mathbb{R}_{\geq 0} \colon R_j(t) \leq r_j$, and
- all jobs are completed: $\forall j \in J \colon \int_0^\infty R_j(t) \, \mathrm{d}t \geq v_j$.

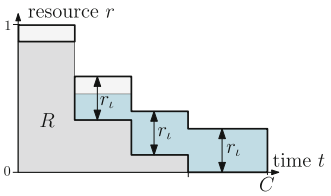
For $j \in J$ we define its *processing time* $p_j := v_j / r_j$ as the minimum time that j requires to be completed. See Fig. 1a for an illustration of these notions.

For a schedule $R = (R_j)_{j \in J}$ we define $C_j(R) := \sup\{t \geq 0 | R_j(t) > 0\}$ as the *completion time* of job $j \in J$. We measure the quality of a schedule R via its *makespan* $M(R) := \max\{C_j(R) | j \in J\}$ and its *total completion time* $C(R) := \sum_{j \in J} C_j(R)$. Our analysis additionally considers the *total fractional completion time* $C^F(R) := \sum_{j \in J} C_j^F(R)$, where $C_j^F(R) := \int_0^\infty R_j(t) \cdot t / v_j \, \mathrm{d}t$ is job j's *fractional completion time*.

Relation to Malleable Tasks with Linear Speedup. Our problem assumes an arbitrarily divisible resource, as for example the bandwidth shared by jobs running



(a) A schedule for six jobs. The resource assignment of job j is given by the height of j's area at time t and must never exceed r_j. The total area of a job is equal to its volume.



(b) Augmenting a schedule R by a job ι via WFstep(R,ι,C). The volume of ι is „poured" into the fat-outlined area. The blue area indicates where it is eventually scheduled.

Fig. 1. Illustration of model notions (left) and a WaterFill step.

on the same host. Another common case are jobs that compete for a *discrete* set of resources, like a number of available processing units. This is typically modeled by a scheduling problem where a set J of n *malleable* jobs of different sizes s_j (length when run on a single machine) must be scheduled on m machines. Each machine can process at most one job per time, but jobs j can be processed on up to $\delta_j \in [m]$ machines in parallel with a linear speedup. Jobs are preemptable, i.e., they can be paused and continued later on, possibly on a different number of machines. See [14, Ch. 25] for a more detailed problem description.

This formulation readily maps to our problem by setting j's processing volume to $v_j = s_j/m$ and its resource requirement to $r_j = \delta_j/m \in (0, 1]$. The only difference is that our schedules allow for arbitrary resource assignments, while malleable job scheduling requires that each job j gets an *integral* number δ_j of machines (i.e., resource assignments must be multiples of $1/m$). However, as observed by Beaumont et al. [3], fractional schedules can be easily transformed to adhere to this constraint:

Observation 1 ([3, Theorem 3, reformulated]). Consider a feasible schedule R for a job set J in which $j \in J$ completes at C_j. Let $m := 1/\min\{r_j | j \in J\}$. We can transform each R_j without changing C_j to get $R_j(t) \in \{i/m | i \in [m] \cup \{0\}\}$ for any $t \in \mathbb{R}_{\geq 0}$ and such that each R_j changes at most once between consecutive completion times.

We first consider *online makespan minimization* (Sect. 2), where the scheduler must commit to future resource assignments as jobs arrive (as in list-scheduling). Afterwards, we consider *offline total completion time minimization* (Sect. 3).

1.2 Related Work

Our model falls into the class of continuous shared-resource job scheduling as introduced in [1] and its variants [5,13]. These models have the same relation between a job's resource requirement, the assigned resource, and the resulting processing time as we but only consider makespan minimization as objective. The two main differences are that they assumed an additional constraint on

the number of machines and considered discrete time slots in which resource assignments may not change.

Another closely related model is *malleable* job scheduling, where the number of machines assigned to a job can be dynamically adjusted over time. If each job j has its own upper limit δ_j on the number of processors it can be assigned, the model becomes basically equivalent to our shared-resource job scheduling problem (as discussed at the end of Sect. 1.1). Drozdowski [7] gave a simple greedy algorithm for minimizing the makespan in the offline setting (see also Sect. 2). Decker et al. [6] considered total completion time minimization for *identical* malleable jobs for an otherwise rather general (possibly non-linear) speed-up function. They gave a 5/4-approximation for this setting. Beaumont et al. [3] is closest to our model. In particular, they assumed job-dependent resource limits δ_j that correspond to our resource requirements. For minimizing weighted total completion time, they used a water-fill approach to prove the existence of structurally nice solutions (cf. to the our water-filling approach in Sect. 2). Their main result is a (non-clairvoyant) 2-approximation algorithm for the weighted case. Their algorithm WDEQ assigns each job a number of processors according to their relative weight, but no more than the limit imposed by δ_j. Our results in Sect. 3 yield an improved approximation ratio of $3/2 + \varepsilon$ at the cost of clairvoyance (i.e., we must know the job's volumes and resource requirements). Also, our algorithm only handles the unweighted case.

Other related models, such as rigid and moldable scheduling, disallow the resource assignment of a job to be adjusted after it has been started (see [14] for details).

1.3 Our Contribution and Methods

For our model, makespan minimization is known to be offline solvable (see Sect. 2). We thus concentrate on an online (list-scheduling) setting where jobs are given sequentially and we must commit to a resource assignment without knowing the number of jobs and future jobs' properties. We use a water-filling approach that is known to produce "flattest" schedules [3]. We derive properties that are necessary and sufficient for any c-competitive algorithm by providing conditions on *c-extendable* schedules (c-competitive schedules to which we can add any job while remaining c-competitive). From this, we derive slightly weaker *universal schedules* that are just barely c-extendable and show that schedules derived via water-fill are always flatter than universal schedules. Optimizing the value of c yields $e/(e-1)$-competitiveness. We then show that no algorithm can have a lower competitive ratio than $e/(e-1)$.

Our main result considers *offline total completion time minimization*. We improve upon the so far best result for this variant (a 2-approximation [3]) by providing a $(3/2 + \varepsilon)$-approximation running polynomial time in $n, 1/\varepsilon$. The result relies on a continuous linear programming (CLP) formulation for the fractional total completion time, for which we consider primal-dual pairs. The primal solution represents the resource assignments over time, while the dual represents the *priority* of jobs over time. We then extract additional properties about the

primal/dual pair. Roughly, our method is as follows. We draw both the primal and dual solutions into a two-dimensional coordinate system. See Fig. 3b for an example. We then merge both solutions into a single 3D coordinate system by sharing the time axis and use the these solutions as a blueprint for shapes in this coordinate system (see Fig. 4). The volume of these shapes then correspond to parts of the primal and dual objective. We use a second algorithm called Greedy that attempts to schedule jobs as early as possible. Choosing the better one of the CLP and the greedy solution gives us the desired approximation.

2 Makespan Minimization

This section considers our resource-aware scheduling problem under the makespan objective. For the offline problem, it is well-known that the optimal makespan $M^*(J)$ for a job set $J = [n]$ with total volume $V(J) = \sum_{j \in J} v_j$ is $M^*(J) = \max\{V(J)\} \cup \{p_j | j \in J\}$ and that a corresponding schedule can be computed in time On [14, Section 25.6]. The idea is to start with a (possibly infeasible) schedule R that finishes all jobs at time $p_{\max} := \max\{p_j | j \in J\}$ by setting $R_j(t) = v_j / p_{\max}$ for $t \in [0, p_{\max})$ and $R_j(t) = 0$ for $t > p_{\max}$. This schedule uses a constant total resource of $\bar{R} := V(J)/p_{\max}$ until all jobs are finished. If $\bar{R} \leq 1$ (the resource is not overused), this schedule is feasible and optimal (any schedule needs time at least $p_{\max}$ to finish the "longest" job). Otherwise we scale all jobs' resource assignments by $1/\bar{R}$ to get a new feasible schedule that uses a constant total resource of 1 until all jobs are finished at time $V(J)$. Again, this is optimal (any schedule needs time at least $V(J)$ to finish a total volume of $V(J)$).

List-Scheduling Setting. Given that the offline problem is easy, the remainder of this section considers the (online) list-scheduling setting. That is, an (online) algorithm $\mathcal{A}$ receives the jobs from $J = [n]$ one after another. Given job $j \in J$, $\mathcal{A}$ must fix j's resource assignment $R_j \colon \mathbb{R}_{\geq 0} \to [0, 1]$ without knowing n or the properties of future jobs. We refer to the resulting schedule by $\mathcal{A}(J)$. As usual in the online setting without full information, we seek to minimize the worst-case ratio between the costs of the computed and optimal schedules. More formally, we say a schedule R for a job set J is *c-competitive* if $M(R) \leq c \cdot M^*(J)$. Similarly, we say an algorithm $\mathcal{A}$ is *c-competitive* if for any job set J we have $M\big(\mathcal{A}(J)\big) \leq c \cdot M^*(J)$.

An Optimal List-Scheduling Algorithm. Water-filling algorithms are natural greedy algorithms for scheduling problems with a continuous, preemptive character. They often yield structurally nice schedules [2–4]. In this section, we show that water-filling (described below) yields a simple, optimal online algorithm for our problem.

Theorem 1. *Algorithm* WaterFill *has competitive ratio $e/(e-1)$ for the makespan. No deterministic online algorithm can have a lower worst-case competitive ratio.*

We first describe a single step WFStep(R, ι, C) of WaterFill (illustrated in Fig. 1b). It takes a schedule $R = (R_j)_{j \in J}$ for some job set J, a new job $\iota \notin J$, and a *target completion time* C. Its goal is to *augment* R *by* ι *with completion time* C, i.e., to feasibly complete ι by time C without altering the resource assignments R_j for any $j \in J$. To this end, define the *h-water-level* $\mathrm{wl}_h(t) := \min\{r_\iota, \max\{h - \bar{R}(t), 0\}\}$ at time t (the resource that can be assigned to ι at time t without exceeding total resource h). Note that ι can be completed by time C iff $\int_0^C \mathrm{wl}_1(t)\,\mathrm{d}t \geq v_\iota$ (the total leftover resource suffices to complete ι's volume by time C). If ι cannot be completed by time C, WFStep(R, ι, C) *fails*. Otherwise, it *succeeds* and returns a schedule that augments R with the resource assignment $R_\iota = \mathrm{wl}_{h^*}$ for job ι, where $h^* := \inf_{h \in [0,1]}\{h | \int_0^C \mathrm{wl}_h(t)\,\mathrm{d}t \geq v_\iota\}$ is the smallest water level at which ι can be scheduled.

WaterFill is defined recursively via WFStep. Given a job set $J = [n]$, define $H_j := M^*([j]) \cdot e/(e-1)$ as the target completion time for job $j \in J$ (remember that $M^*([j])$ can be easily computed, as described at the beginning of this section). Assuming WaterFill computed a feasible schedule $R^{(j-1)}$ for the first $j-1$ jobs (with $R^{(0)}(t) = 0\ \forall t \in \mathbb{R}_{\geq 0}$), we set $R^{(j)} :=$ WFStep$(R^{(j-1)}, j, H_j)$. If this step succeeds, the resulting schedule is clearly $e/(e-1)$-competitive by the choice of H_j. The key part of the analysis is to show that indeed these water-filling steps always succeed.

We start the observation that water-fill schedules always result in "staircase-like" schedules (see Fig. 1b), a fact also stated in [3] (using a slightly different wording).

Observation 2 ([3, Lemma 3]). Consider a schedule R whose total resource usage $\bar{R}$ is non-increasing (piecewise constant). If we WFStep(R, ι, C) successfully augments R by a job ι, the resulting total resource usage is also non-increasing (piecewise constant).

Next, we formalize that WFStep generates the "flattest" schedules: if there is *some* way to augment a schedule by a job that completes until time C, then the augmentation can be done via WFStep.

Definition 1. The *upper resource distribution* $A_R^C(y)$ of a schedule R is the total volume above height y before time C in R. Given schedules R, S (for possibly different job sets), we say R is *flatter* than S ($R \preceq S$) if $A_R^C(y) \leq A_S^C(y)$ $\forall C \in \mathbb{R}_{\geq 0}, y \in [0,1]$.

Lemma 1 ([3, Lemma 4, slightly generalized]). *Consider two schedules $R \preceq S$ for possibly different job sets. Let S' denote a valid schedule that augments S by a new job ι completed until time C. Then* WFStep(R, ι, C) *succeeds and* WFStep$(R, \iota, C) \preceq S'$.

Next, we characterize c-competitive schedules that can be augmented by *any* job while staying c-competitive.

Definition 2. A schedule R is *c-extendable* if it is c-competitive and if it can be feasibly augmented by *any* new job ι such that the resulting schedule is also c-competitive.

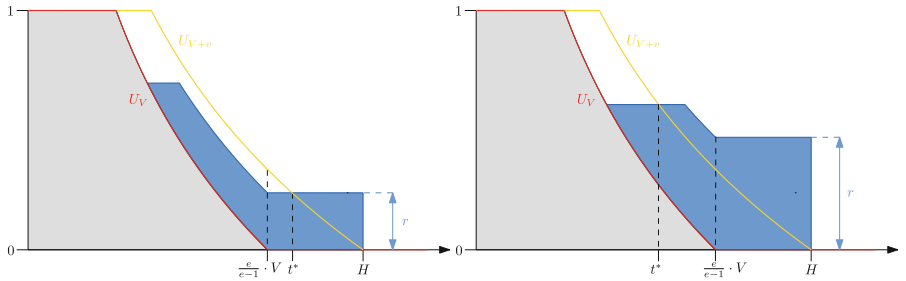
Lemma 2. *Consider a job set J of volume V and with maximal processing time $p_{\max}$. A c-competitive schedule R for J is c-extendable if and only if*

$$\forall y \text{ with } (c-1)/c < y \leq 1\colon \quad A_R^{\infty}(y) \leq (c-1)\cdot(1-y)/y \cdot \max\{V, p_{\max}\cdot y\}. \quad (1)$$

See the Full Version for the proof of Lemma 2. While Lemma 2 gives a strong characterization, the bound on the right hand side of Eq. (1) cannot be easily translated into a proper schedule for the given volume. Thus we introduce proper (idealized) schedules that adhere to a slightly weaker version of Eq. (1). These schedules are barely $e/(e-1)$-extendable. Our proof of Theorem 1 combines their existence with Lemma 1 to deduce that WaterFill is $e/(e-1)$-competitive.

Definition 3. For any $V \in \mathbb{R}_{\geq 0}$ we define the *universal schedule*[1] $U_V\colon \mathbb{R}_{\geq 0} \to [0,1]$ via

$$U_V(t) := \begin{cases} 1 & \text{if } t < \frac{1}{e-1}\cdot V, \\ 1 - \ln\left(t \cdot \frac{c-1}{V}\right) & \text{if } \frac{1}{e-1}\cdot V \leq t < \frac{e}{e-1}\cdot V, \text{and} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$



(a) U_{V+v} intersects W at the lower plateau. (b) U_{V+v} intersects W at the upper plateau.

Fig. 2. Universal schedules U_V and U_{V+v}. The blue area indicates a new job ι with volume v and resource requirement r that is scheduled via WFstep(U_V, ι, H). Depending on the resource requirement r, the yellow line enters the blue area exactly once, either on the upper plateau (a) or on the lower plateau (b).

See Fig. 2 for an illustration of universal schedules. With $c = e/(e-1)$, one can easily check that $A_{U_V}^{\infty}(y) = \frac{e^{1-y}-1}{e-1}\cdot V \leq (c-1)\cdot\frac{1-y}{y}\cdot V$. Thus, by Lemma 2, universal schedules (and any flatter schedules for the same volume) are $e/(e-1)$-extendable. Our final auxiliary lemma extends the optimality of WaterFill from Lemma 1 to certain augmentations of universal schedules.[2] See the Full Version for the proof of Lemma 3.

[1] One can think of U_V as a schedule for a single job of volume V and resource requirement 1. Since there is only one job, we identify U_V with its total resource requirement function $\bar{U}_V$.

[2] Lemma 3 is not a special case of Lemma 1: the schedule S' from Lemma 1 must adhere to the new job's resource requirement, which is not the case for the universal schedule U_{V+v}.

Lemma 3. *Consider the universal schedule U_V, a new job ι of volume v and processing time p, as well as a target completion time $H \geq \frac{e}{e-1} \cdot \max\{V+v, p\}$. Then* $\text{WFSTEP}(U_V, \iota, H) \preceq U_{V+v}$.

The above enables us to prove the competitiveness of WATERFILL from Theorem 1: We show inductively that WATERFILL produces a feasible schedule $R^{(j)}$ for the first j jobs (using that $R^{(j-1)}$ is "flatter" than $U_{V([j-1])}$ together with Lemma 1) and use this to prove $R^{(j)} \preceq U_{V([j])}$ (via Lemma 3). By universality, this implies that all $R^{(j)}$ are $e/(e-1)$-extendable (and thus, in particular, $e/(e-1)$-competitive). The full proof of WATERFILL is given in the Full Version.

3 Total Completion Time Minimization

This section considers the total completion time minimization and represents our main contribution. In contrast to offline makespan minimization (Sect. 2), it remains unknown whether there is an efficient algorithm to compute an offline schedule with minimal total completion time. The so far best polynomial-time algorithm achieved a 2-approximation [3]. We improve upon this, as stated in the following theorem.

Theorem 2. *There is a $(3/2+\varepsilon)$-approximation algorithm for total completion time minimization. Its running time in polynomial time in n and $1/\varepsilon$.*

For clarity of presentation, we analyze an idealized setting in the main part. The details for the actual result can be found in the Full Version.

Algorithm Description. Our algorithm computes two *candidate schedules* using the two sub-algorithms GREEDY and LSAPPROX (described below). It then returns the schedule with smallest total completion time among both candidates.

Sub-algorithm GREEDY processes the jobs in ascending order of their volume. To process a job, GREEDY assigns it as much resource as possible as early as possible in the schedule. Formally, for jobs $J = [n]$ ordered as $v_1 \leq \cdots \leq v_n$, the schedule R^G for GREEDY is calculated recursively using $R^G_j(t) = \mathbb{1}_{t<t_j} \cdot \min(r_j, 1 - \sum_{i=1}^{j-1} R^G_i(t))$, where the completion time t_j for job j is set such that j schedules exactly its volume v_j. See Fig. 3b for an example of a GREEDY schedule. Sub-algorithm LSAPPROX deals with solutions to following *continuous linear program* (CLP).

$$\text{minimize} \sum_{j \in J} \int_0^\infty \frac{t \cdot R_j(t)}{v_j} \,\mathrm{d}t \qquad \int_0^\infty R_j(t) \,\mathrm{d}t \geq v_j \quad \forall j \in J$$

$$0 \leq R_j(t) \leq r_j \quad \forall j \in J, t \in \mathbb{R}_{\geq 0} \qquad \sum\nolimits_{j \in J} R_j(t) \leq 1 \quad \forall t \in \mathbb{R}_{\geq 0}$$

Roughly, LSAPPROX first subdivides the job set into those jobs that produce a high completion time and the remaining jobs. For the former, an approximate solution is computed using the dual to the discretization (an LP) of above CLP. For the latter, is enough to reserve a small portion of the resource to schedule

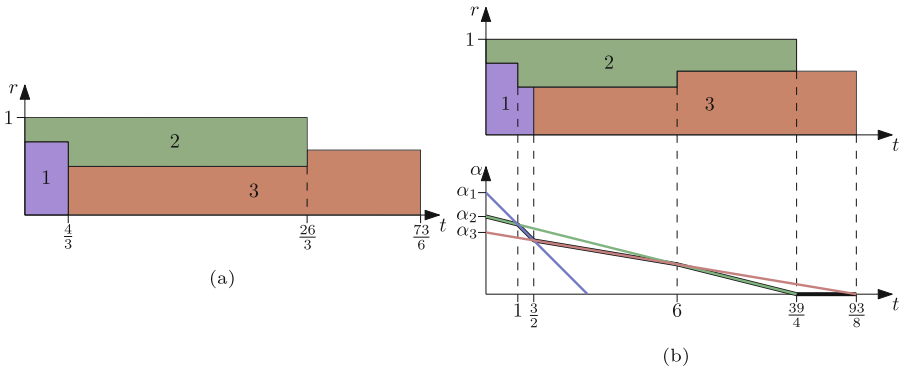


Fig. 3. Schedules for a job set $J = [3]$ with $(v_1, r_1) = (1, 3/4)$, $(v_2, r_2) = (4, 1/2)$ and $(v_3, r_3) = (6, 2/3)$. (a) Greedy's schedule, (b) Above: A primal (resource) schedule. Below: A dual (priority) schedule. With the dual variables having values $\alpha_1 = 51/16$,$\alpha_2 = 39/16$ and $\alpha_3 = 31/16$, the volumes of the jobs are exactly scheduled. (See the Full Version.)

them with small completion times. For clarity of presentation, the main part will only do a simplified analysis using an idealization of LSApprox. For the details of this algorithm, the analysis using LSApprox and the analysis of Greedy, we refer to the Full Version.

3.1 Analysis via a Bounded Fractionality Gap

Throughout the analysis, we use C^* to denote the optimal total completion time and C^{F*} for the optimal fractional total completion time. We require an algorithm that produces a schedule R with a small *fractionality gap* $\gamma(R) := C(R)/C^{F*}$, i.e., we compare the total completion time of R with the optimal fractional total completion time for the same job set. We show the following generalization of Theorem 2.

Theorem 3. *Assume that there is a polynomial-time algorithm A for total completion time minimization that produces a schedule R with $\gamma(R) \geq 1$. Then there exists a polynomial-time $(\gamma(R) + 1)/2$-approximation for total completion time minimization.*

The proof of Theorem 3 relies on Proposition 1 (three lower bounds on the optimal total completion time) and Proposition 2 (Greedy's objective in relation to these bounds). Lower Bound (1) (*Squashed Area* Bound) and Bound (2) (*Length* or *Height* Bound) are due to Beaumont et al. [3, Def. 6,7]. Bound (3) is our novel lower bound. The proof can be found in the Full Version.

Proposition 1. *Assuming $v_1 \leq \cdots \leq v_n$, the following are lower bounds on C^*:*

$$(1)\ C^L := \max_{j\in J} p_j \qquad (2)\ C^A := \sum_{j=1}^{n}\sum_{i=1}^{j} v_j \qquad (3)\ C^{F*} + 1/2 \cdot C^L$$

Proposition 2. *The* Greedy *schedule* R^G *satisfies* $C(R^G) \le C^A + C^L$.

Using them, we can give the proof of Theorem 3.

Proof of Theorem 3. We run both Greedy and A in polynomial time to produce schedules R^G and R^A, respectively, and choose the schedule with the smaller total completion time. Using Proposition 1 and 2 and the fractionality gap $\gamma := \gamma(R^A)$, we can bound the cost $C := \min(C(R^A), C(R^G))$ of the resulting schedule in terms of C^*:

$$C \le \min(\gamma \cdot C^{F*}, C^A + C^L) \le \min(\gamma \cdot (C^* - 1/2 \cdot C^L), C^* + C^L)$$
$$= \frac{\gamma+1}{2}C^* - \frac{\gamma+2}{4}C^L + \min\left(\frac{\gamma-1}{2}C^* - \frac{\gamma+2}{4}C^L, \frac{\gamma+2}{4}C^L - \frac{\gamma-1}{2}C^*\right) \le \frac{\gamma+1}{2}C^*$$

□

3.2 The Fractionality Gap of Line Schedules

For the remainder of this paper, we will introduce *line schedules* and their structural properties. Roughly, a line schedule is a certain primal-dual pair for the CLP defined in Sect. 3, and its dual, which we call DCP:

$$\begin{aligned} \text{maximize } & \sum_{j\in J} \alpha_j v_j - \sum_{j\in J} r_j \int_0^\infty \beta_j(t)\,\mathrm{d}t - \int_0^\infty \gamma(t)\,\mathrm{d}t \\ \text{s.t. } & \alpha_j, \beta_j(t), \gamma(t) \ge 0 \quad \forall j \in J, t \in \mathbb{R}_{\ge 0} \qquad \gamma(t) + \beta_j(t) \ge \alpha_j - t/v_j \quad \forall j \in J, t \in \mathbb{R}_{\ge 0} \end{aligned}$$

It is obtained by dualizing the time-discretized version of the CLP (see the Full Version) and extending its constraints to the continuous time domain. *Line schedules* formalize the idea that, if we know the dual α-values, we can reconstruct all remaining primal/dual variables to obtain a primal-dual pair. If the α-values are chosen correctly, then the volumes scheduled in the primal are exactly the desired volumes $(v_j)_{j\in J}$.

To this end, we will *assume* that we have access to an algorithm called LS that produces such a line schedule R^F with $C^F(R^F) = C^{F*}$. We can then show that LS produces schedules with a fractionality gap of 2:

Proposition 3. *The* LS *schedule* R^F *satisfies* $\gamma(R^F) \le 2$.

In the following, we develop the details of line schedules. To this end, first define *primal-dual pair* as a tuple $(R, \alpha, \beta, \gamma, v)$ that fulfills the following continuous *slackness conditions (sc)*. Again, these are found by extending the time-discretized version of the CLP to the continuous time domain. These conditions hold for all $j \in J$ and $t \in \mathbb{R}_{\ge 0}$.

$$(\alpha\text{-sc}): \alpha_j\left(\bar{v}_j - \int_0^\infty R_j(t)\,\mathrm{d}t\right) = 0 \qquad (\beta\text{-sc}): \beta_j(t)(r_j - R_j(t)) = 0$$
$$(\gamma\text{-sc}): \gamma(t)\left(1 - \sum_{j\in J} R_j(t)\right) = 0 \qquad (R\text{-sc}): R_j(t)(\alpha_j - t/v_j - \beta_j(t) - \gamma(t)) = 0$$

If we choose arbitrary α-values, then the corresponding line schedule is still a primal-dual pair, except that it possibly schedules a different set of volumes, i.e., the α-sc is only true if we replace v_j in the constraint by some other volume $\bar{v}_j$. This fact is used for the detailed proof of our $(3/2+\varepsilon)$-approximation, see the Full Version.

To this end, define the *dual line* $d_j(t) := \alpha_j - t/v_j$ for each $j \in J$. The intuition behind a line schedule is now that the heights of the dual lines represent priorities: Jobs are scheduled (with maximum remaining schedulable resource) in decreasing order of the dual line heights at the respective time points. Jobs are not scheduled if their dual line lies below zero. This is formalized in the following definition. (In Fig. 3b, we supplement the example from Fig. 3a by a depiction of the dual lines.)

Definition 4. We call a job set J *non-degenerate* if all job volumes are pairwise distinct, i.e., $v_j \neq v_{j'}$ for all $j, j' \in J$.[3] Define a total order for each $t \geq 0$ as $j' \succ_t j :\Leftrightarrow d_{j'}(t) > d_j(t)$ or $d_{j'}(t) = d_j(t)$ and $v_{j'} > v_j$.[4] The *line schedule* of α is a tuple $(R, \alpha, \beta, \gamma, v)$ (recursively) defined as follows.

$$R_j(t) = \mathbb{1}_{d_j(t)>0} \cdot \min(r_j, 1 - \sum\nolimits_{j' \succ_t j} R_{j'}(t)) \qquad \beta_j(t) = \max(0, d_j(t) - \gamma(t))$$

$$\gamma(t) = \max(0, d_j(t)), \text{ where } j \text{ is the smallest job according to } \succ_t \text{ with } R_j(t) > 0$$

Equipped with the definition of a line schedule, we can now tackle the proof of Proposition 3. It requires the following two properties about the assumed algorithm LS. First, Lemma 4 allows us to bound the completion times of a fractional schedule in terms of the α-variables in the DCP:

Lemma 4. *Algorithm LS produces a schedule R^F with $C_j(R^F) \leq \alpha_j v_j$ for all $j \in J$.*

Second, we show the following lemma. Abbreviate $P = \sum_{j \in J} \int_0^\infty t \cdot R_j(t)/v_j \, \mathrm{d}t$ (the primals objective), and $A = \sum_{j \in J} \alpha_j v_j$, $B = \sum_{j \in J} r_j \int_0^\infty \beta_j(t) \, \mathrm{d}t$ and $\Gamma = \int_0^\infty \gamma(t) \, \mathrm{d}t$ (the parts of the dual objective).

Lemma 5. *Algorithm LS produces a schedule R^F such that there exists a primal-dual pair $(R^F, \cdot, \cdot, \cdot)$ that fulfills strong duality ($A = B + \Gamma + P$) and balancedness ($P = B + \Gamma$).*

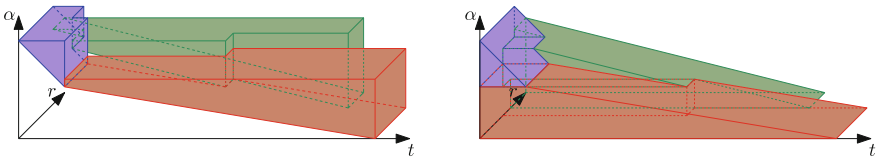
Using these lemmas, we can show Proposition 3.

Proof of Theorem 3. Using Lemmas 4 and 5, we show the statement as follows:

$$C(R^F) = \sum_{j \in J} C_j(R^F) \leq \sum_{j \in J} \alpha_j v_j = A = A - B - \Gamma + P = 2P = 2C^F(R^F) = 2C^{F*}$$

[3] While not strictly required, this makes line schedules unique and simplifies the analysis.

[4] The second part of the definition ($d_{j'}(t) = d_j(t)$ and $v_{j'} > v_j$) only exists for disambiguation of the line schedule, but is not further relevant.



(a) (b)

Fig. 4. (a) P-shapes for job set from Fig. 3b. P-shapes are delimited from below by $d_j(t)$ (extended into the resource axis), from above by α_j, and their top surface follows the primal schedule. (b) The shapes shown represent the union of B- and Γ-shapes. They are delimited from the left (right) by $t = 0$ ($d_j(t)$) (extended into the resource axis), and from top and bottom by the value of $d_j(t)$ at the starting and finishing time of some piece of j. See the Full Version for the formal definition of these shapes.

□

In the Full Version, we show the following Lemma 6, stating that line schedules are indeed primal-dual pairs. We then define LS to output a schedule R^F for a line schedule $(R^F, \alpha, \beta, \gamma, v)$ according to Lemma 6, i.e., for each $j \in J$, $\int_0^\infty R_j(t)\,\mathrm{d}t = v_j$. Using this definition, we can show Lemma 4.

Lemma 6. *For any job set J there exists an α such that the line schedule of α is a primal-dual pair.*

Proof of Lemma 4. By definition, $R_j^F(t) = 0$ if $d_j(t) \leq 0$. Hence, as d_j is monotonically decreasing, $C_j(R^F)$ is bounded by the zero of $d_j(t)$, which lies at $t = \alpha_j v_j$. □

The remainder of this section will initiate the proof of Lemma 5. We first give a geometric understanding of the involved quantities (P, A, B, Γ). We build a 3D coordinate system from a line schedule. The time axis is shared, and the ordinates form the remaining two axes. We then draw 3D shapes into this coordinate system that correspond to parts of the above quantities and therefore of the CLP/DCP objectives. These shapes are described in detail in the Full Version. Generally, these shapes are constructed such that the primal and dual schedules can be "seen" from above or front. In our case, the primal schedule will be seen from the top, and the dual schedule from the front. Figure 4 illustrates the shapes in our construction. For each part of the objective $\Psi \in \{P, A, B, \Gamma\}$, we have a corresponding shape Ψ^{all}, which is subdivided into pieces $\Psi^{i,l}$, respectively.

We can show that certain pieces are pairwise non-overlapping (Lemma 7), that the A-pieces make up all other pieces (Lemma 8) and we can relate the volume of these pieces with one another and with the actual objective (Lemma 9).

Lemma 7. *Let V and W, $V \neq W$, be P-pieces, B-pieces or Γ-pieces (every combination allowed), or both be A-pieces. Then V and W do not overlap.*

Lemma 8. *A^{all} is composed of the other shapes, i.e., $A^{\text{all}} = P^{\text{all}} \cup B^{\text{all}} \cup \Gamma^{\text{all}}$.*

Lemma 9. *The pieces satisfy* $|P^{i,l}| = |B^{i,l}| + |\Gamma^{i,l}|$ *for all* i, l *and* $|\Psi^{\text{all}}| = \Psi$ *for all* $\Psi \in \{P, A, B, \Gamma\}$.

Due to space limitations, we give the actual construction of the pieces and the proofs of Lemma 7 to 9 in the Full Version. Now we can give the proof of Lemma 5.

Proof of Lemma 5. Using Lemma 7 to 9, we get

$$A = |A^{\text{all}}| = |P^{\text{all}} \cup B^{\text{all}} \cup \Gamma^{\text{all}}| = |P^{\text{all}}| + |B^{\text{all}}| + |\Gamma^{\text{all}}| = P + B + \Gamma$$

$$= |P^{\text{all}}| + |B^{\text{all}}| + |\Gamma^{\text{all}}| = \sum\nolimits_{i,l} |P^{i,l}| + |B^{i,l}| + |\Gamma^{i,l}| = \sum\nolimits_{i,l} 2|P^{i,l}| = 2P.$$

□

References

1. Althaus, E., et al.: Scheduling shared continuous resources on many-cores. J. Sched. **21**(1), 77–92 (2018). https://doi.org/10.1007/s10951-017-0518-0
2. Antoniadis, A., Kling, P., Ott, S., Riechers, S.: Continuous speed scaling with variability: a simple and direct approach. Theor. Comput. Sci. **678**, 1–13 (2017)
3. Beaumont, O., Bonichon, N., Eyraud-Dubois, L., Marchal, L.: Minimizing weighted mean completion time for malleable tasks scheduling. In: IEEE 26th IPDPS, pp. 273–284 (2012)
4. Chen, D., Wu, J., Liu, P.: Data-bandwidth-aware job scheduling in grid and cluster environments. In: 15th IEEE ICPADS, pp. 414–421 (2009)
5. Damerius, C., Kling, P., Li, M., Schneider, F., Zhang, R.: Improved scheduling with a shared resource via structural insights. In: COCOA, pp. 168–182 (2020)
6. Decker, T., Lücking, T., Monien, B.: A 5/4-approximation algorithm for scheduling identical malleable tasks. TCS **361**(2–3), 226–240 (2006)
7. Drozdowski, M.: New applications of the Muntz and Coffman algorithm. J. Sched. **4**(4), 209–223 (2001)
8. Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput. **4**(4), 397–411 (1975)
9. Grigoriev, A., Sviridenko, M., Uetz, M.: Machine scheduling with resource dependent processing times. Math. Program. **110**(1), 209–228 (2007)
10. Grigoriev, A., Uetz, M.: Scheduling jobs with time-resource tradeoff via nonlinear programming. Discret. Optim. **6**(4), 414–419 (2009)
11. Jansen, K., Maack, M., Rau, M.: Approximation schemes for machine scheduling with resource (in-)dependent processing times. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM SODA, pp. 1526–1542 (2016)
12. Kellerer, H.: An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. Oper. Res. Lett. **36**(2), 157–159 (2008). https://doi.org/10.1016/j.orl.2007.08.001
13. Kling, P., Mäcker, A., Riechers, S., Skopalik, A.: Sharing is caring: multiprocessor scheduling with a sharable resource. In: Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA, pp. 123–132. ACM (2017)
14. Leung, J.Y. (ed.): Handbook of Scheduling - Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC, Boca Raton (2004). ISBN 978-1-58488-397-5

15. Maack, M., Pukrop, S., Rasmussen, A.R.: (in-)approximability results for interval, resource restricted, and low rank scheduling. In: 30th Annual European Symposium on Algorithms, ESA, LIPIcs, vol. 244, pp. 77:1–77:13 (2022)
16. Niemeier, M., Wiese, A.: Scheduling with an orthogonal resource constraint. Algorithmica **71**(4), 837–858 (2015). https://doi.org/10.1007/s00453-013-9829-5