Near-Optimal Universal Scheduling for Moldable Tasks: The FAIR Algorithm

Lucas Perotin*
Thomas Verrecchia
Padma Raghavan*

May 7, 2025

Abstract

The problem of scheduling moldable tasks on multiprocessor systems with the goal of minimizing the makespan has received significant attention, especially in contexts where tasks have dependence constraints (task graphs) or arrive dynamically (online scheduling). However, the combined setting — online scheduling of moldable task graphs — remains less explored.

Prior studies have addressed this problem by designing algorithms specifically tailored to particular speedup models, such as Amdahl, Communication, or Roofline. Although these specialized algorithms achieve constant competitive ratios under their respective models, their applicability is severely limited in practice: each requires tasks to strictly adhere to a predefined speedup model, making them impractical for general scenarios or hybrid environments.

In this paper, we introduce a novel, universal online algorithm capable of handling arbitrary moldable tasks without assumptions about their speedup behavior. Remarkably, when applied to the specific speedup models previously studied, our algorithm achieves competitive ratios comparable to — or better than — those obtained by the specialized algorithms. Extensive experiments confirm that our universal approach consistently matches or outperforms existing model-specific algorithms, thus bridging the gap between theoretical guarantees and practical applicability.

1 Introduction

This paper addresses the problem of online scheduling of moldable task graphs onto a set of identical processors. Moldable tasks constitute an important and flexible class of parallel tasks where each task can run on an arbitrary number of processors, determined at the start of its execution, and the allocation remains fixed throughout. The execution time t of a moldable task directly depends on the number of processors p allocated to it, which is represented by a speedup function $t(p)$. This static yet flexible allocation strategy contrasts notably with rigid tasks, whose processor allocation is fixed and predefined, and malleable tasks, whose processor allocation can vary dynamically during execution [9].

Moldable tasks strike an appealing balance between flexibility and practicality: unlike rigid tasks, they effectively exploit available computational resources, and unlike malleable tasks, they avoid the significant complexities and overheads associated with dynamic resource reallocation. Consequently, moldable tasks are extensively employed in practice, especially in scientific computing contexts like numerical linear algebra or tensor operations, where computational kernels are frequently implemented in a moldable form. In this work, we consider scheduling to be non-preemptive and without restarts, avoiding overheads due to checkpointing, context switching, and task migration [10].

Given their practical significance, moldable tasks have been extensively studied within the scheduling community, including in online scheduling scenarios, where tasks become available only when they are ready to execute (i.e., after all their predecessors have completed), and their characteristics remain unknown until that moment. Online scheduling algorithms aim to minimize the makespan — the overall execution time — despite incomplete knowledge about future tasks and their dependencies. To evaluate the performance of online scheduling algorithms, one typically compares them against an optimal offline scheduler, leading to the notion of competitive ratio: a metric capturing how closely an online algorithm approximates the best offline solution under the worst-case adversarial scenario [22].

*Vanderbilt University, lucas.perotin@vanderbilt.edu

Previous work [5, 21] in this area has primarily focused on designing specialized online algorithms tailored to specific speedup models, such as the Roofline, Communication, or Amdahl models, which will be defined later. While these specialized algorithms achieve strong theoretical guarantees with constant competitive ratios, their practical use is severely restricted since each algorithm presumes tasks adhere strictly to one predefined speedup model. Such constraints rarely, if ever, reflect real-world conditions.

In this paper, we propose a novel universal online algorithm, Fair, that effectively schedules moldable tasks without requiring adherence to any particular speedup model. The primary goal of our approach is not to achieve superior competitive ratios for specific scenarios, but rather to offer a robust and universally applicable method across diverse task behaviors. Nonetheless, our universal algorithm achieves competitive ratios closely matching — and in some cases surpassing — those obtained by existing specialized solutions, as illustrated in Table 1. To achieve these results, we introduce a refined analytical framework, incorporating sharper bounds and novel techniques tailored to our heuristic in the moldable task setting.

Table 1: Competitive ratios (rounded up to the nearest hundredth) for different speedup models

Algorithm	Roofline	Communication	Amdahl	General
[5]	2.62	3.61	4.74	5.72
[21]	2.62	3.40	4.55	4.63
[This Paper]	2.62	3.41	4.56	4.60

Our simplified general result is as follows. **If for each task in the instance, there exists a processor allocation such that both the execution time and the area (defined as the product of execution time and number of processors) are within a factor R of their respective minima, then the schedule produced by Fair achieves a makespan within a factor $\frac{2R+1+\sqrt{4R^2+1}}{2} < 2R+1$ of the optimal clairvoyant schedule**. A similar competitive ratio of $2R$ was achieved by Lpa-List [3, 4], but in a much simpler setting consisting of independent tasks subject to failures, a scenario that can be interpreted as an online graph composed of chains of identical tasks. Our result, very similar, applies to the significantly more general and challenging setting of arbitrary moldable task graphs.

Furthermore, for arbitrary speedup models without superlinear speedups, our algorithm achieves a competitive ratio of $2\sqrt{P}+3$, where P is the number of processors, which we prove to be asymptotically optimal. To justify this, we present a new lower bound analysis based on an instance with $n = P$ tasks. In this work, n always represents the number of tasks in the task graph. This instance adapts to the decisions of any online algorithm and shows that no competitive ratio better than $\Theta(\sqrt{P})$ or $\Theta(\sqrt{n})$ is achievable. Prior work [21] established only weaker results, providing a looser lower bound of $\Theta(\ln(\ln(n)))$. Our paper not only introduces a highly practical universal algorithm but also advances the understanding of the best possible competitive ratio achievable in online moldable task scheduling.

Finally, extensive experimental evaluations using tens of thousands of synthetic task graphs, covering diverse task and graph types, highlight the practical efficiency of our universal algorithm. On average across all scenarios, our algorithm performs similarly or better than previous model-specific algorithms, achieving an average relative performance ratio of 1.52, coincidentally matching the state-of-the-art algorithm presented in [21], and achieving better results than the one introduced in [5]. These results confirm the practical relevance and theoretical robustness of our universal approach.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the formal model, the problem statement, and summarizes our main results. In Section 4, we present Fair, a new online scheduling algorithm. Its general competitive ratio is established in Section 5, where we also prove its near-optimality for arbitrary speedup profiles and task graphs. Section 6 explores this general result under specific speedup models, and argues why significant improvements are unlikely for any of these restricted settings. Section 7 provides a comparison of Fair against previous heuristics through extensive synthetic experiments. Finally, Section 8 concludes the paper and outlines future research directions.

2 Related Work

Numerous studies have explored the offline scheduling of independent moldable tasks, often focusing on achieving approximation guarantees. Some of these works assume specific speedup models, while others tackle the more general setting. Turek et al. [23] proposed a 2-approximation list-based algorithm for the arbitrary speedup model. In a variant where each task can only be assigned a limited set of processor counts, Jansen [14] pro-

Table 2: Problem types in moldable task scheduling.

Problem Type	Offline	Online
Independent moldable tasks	[23, 14, 13]	[26, 12, 8, 19]
Moldable task graphs	[24, 7, 20, 16]	[10, 5, 21], [This paper]

vided a $(1.5 + \epsilon)$-approximation algorithm, which is optimal under standard complexity assumptions [17]. In the monotonic model—where execution time is non-increasing and area (the product of processors and time) is non-decreasing—Jansen and Land [13] developed a polynomial-time approximation scheme (PTAS).

In the online setting with independent moldable tasks arriving over time, Ye et al. [26] devised a 16.74-competitive algorithm. They also described a reduction that transforms any ρ-competitive algorithm for rigid tasks into a 4ρ-competitive algorithm for moldable tasks. Additionally, some offline strategies that schedule tasks independently can naturally extend to the online setting, such as those proposed in [12, 8, 19] under communication-based models.

For moldable tasks with dependencies in the offline context, Wang and Cheng [24] showed that the Earliest Completion Time policy gives a $(3 - 2/P)$-approximation in the roofline model, where P denotes the number of available processors. Lepère et al. [20] achieved a $3 + \sqrt{5}$ approximation for the monotonic model, later improved to 4.73 by Jansen and Zhang [16]. Chen and Chu [7] refined these bounds under stricter assumptions, specifically when area is a concave function and execution time strictly decreases with processor count.

In the online setting with dependent moldable tasks, Feldmann et al. [10] proposed a 2.618-competitive algorithm under the roofline model. Their method maintains high system utilization through adaptive bounds, even without prior knowledge of task durations or graph structure. Canon et al. [6] examined heterogeneous platforms with multiple processor types (e.g., CPUs and GPUs), deriving competitive bounds based on resource heterogeneity, though their work did not extend to moldable tasks.

More recently, Benoit et al. [3, 4] studied independent moldable tasks subject to failures, where a task must be re-executed until it completes successfully. Although all tasks are known in advance, failures are only revealed upon execution, yielding a semi-online model. We do not address task failures in this paper, but rather focus on the general online scheduling of moldable task graphs (as in [10, 5, 21]). However, our results can readily carry over to the failure scenario.

A summary of various problem settings and a non-exhaustive selection of corresponding prior works is presented in Table 2.

3 Problem Statement

In this section, we formally present the online scheduling model and the objective function. We also show a simple lower bound on the optimal makespan, against which the performance of our online algorithms will be measured.

3.1 Model and Objective

We study the online scheduling of a ***Directed Acyclic Graph (DAG)*** composed of moldable tasks on a system with P identical processors. Let $G = (V, E)$ denote the task graph, where $V = \{1, 2, \dots, n\}$ is the set of n tasks and $E \subseteq V \times V$ encodes the precedence constraints between them. An edge $(i, j) \in E$ signifies that task j cannot begin execution until task i has completed. In such a case, task i is referred to as a ***predecessor*** of task j, and task j as a ***successor*** of task i. We do not account for data transfer costs between dependent tasks in this model.

Each task is ***moldable***, meaning that the scheduler determines the number of processors assigned to a task at its start, but this allocation remains fixed throughout its execution. The execution time $t_j(p_j)$ of task j depends on the number p_j of processors assigned, where p_j must be an integer in the range $[1, P]$. In this work, we focus on the following forms of execution time functions:

- ***Roofline Model*** [25]:

$$t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)} \ . \tag{1}$$

  This model assumes ideal linear speedup up to a saturation threshold $\bar{p}_j \leq P$, beyond which allocating additional processors provides no further acceleration. Here, w_j denotes the total work of the task.

- ***Communication Model*** [12]:

$$t_j(p_j) = \frac{w_j}{p_j} + c_j(p_j - 1) \ . \tag{2}$$

In this model, the task is fully parallelizable, but incurs a linear communication cost as more processors are used. The parameter c_j models the per-processor communication overhead.

- ***Amdahl's Model*** [2]:

$$t_j(p_j) = \frac{w_j}{p_j} + d_j \ . \tag{3}$$

This model separates the task into a parallelizable portion of work w_j and a sequential part of fixed cost d_j. The execution time decreases with more processors, but is lower-bounded by the sequential portion.

- ***General Model***:

$$t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)} + d_j + c_j(p_j - 1) \ , \tag{4}$$

This model extends all previous ones by including a maximum parallelism threshold $\bar{p}_j$, a sequential portion d_j, and a linear communication cost c_j. It generalizes the Roofline, Communication, and Amdahl models depending on the values of its parameters.

- ***Power Communication Model***:

$$t_j(p_j) = \frac{w_j}{p_j} + c_j \cdot p_j^{\gamma_j} \ . \tag{5}$$

Unlike the linear Communication Model, this version allows the communication overhead to follow a sublinear or linear growth, depending on the exponent $\gamma_j \in [0, 1]$. When $\gamma_j = 0$, it reduces to Amdahl's Model; when $\gamma_j = 1$, it coincides with the Communication Model.

We assume throughout this work that all parameters satisfy $w_j \geq 0$, $c_j \geq 0$, and $d_j \geq 0$.

Definition 1. *Given the execution time function $t_j(p_j)$ of a task j, we define its* **area** *as a function of the number of allocated processors:*

$$a_j(p_j) = p_j \cdot t_j(p_j).$$

Intuitively, this quantity captures the total processor-time consumed during the execution of task j.

In this work, we adopt the ***online scheduling*** framework, where a task is released only once all of its predecessors have completed. This setting models ***dynamic task graphs***, in which the structure of dependencies unfolds progressively during execution [18, 10, 1, 6].

When a task j becomes ready, its associated execution parameters (such as w_j, $\bar{p}_j$, d_j, c_j, or γ_j) are revealed to the scheduler. The objective is to compute a feasible schedule that minimizes the overall completion time (or ***makespan***), denoted by T. As in many other works on this model (see most papers in the related-work section), we assume that scheduling decisions and task launches occur instantaneously, as their durations are negligible compared to task execution times.

The performance of an online scheduling algorithm is measured by its ***competitive ratio***. An algorithm is said to be c-***competitive*** if, for any instance of the problem, the makespan T it produces satisfies:

$$\frac{T}{T^{\text{opt}}} \leq c,$$

where T^{opt} denotes the optimal makespan achievable by an offline algorithm that has complete prior knowledge of all tasks, their characteristics, and the full dependency graph. The competitive analysis assumes a worst-case perspective, in which an adversary may design inputs that aim to challenge the online algorithm's decisions.

3.2 Assumptions on Speedups

For any task j, we define $p_j^{\max}$ as the processor allocation that minimizes its execution time. We assume that tasks are ***monotonic*** on the interval $[1, p_j^{\max}]$, as formalized below.

Definition 2. *A task satisfies the* **monotonic** *property [20, 5] if the following two conditions hold:*

- *The execution time is a* **non-increasing** *function of the processor allocation up to the optimal point, i.e. $t_j(p) \geq t_j(q)$ for all $1 \leq p < q \leq p_j^{\max}$.*
- *The area is a* **non-decreasing** *function of the processor allocation, i.e. $a_j(p) \leq a_j(q)$ for all $1 \leq p < q \leq p_j^{\max}$.*

The second condition immediately implies that a task cannot achieve super-linear speedup:

$$\frac{t_j(p)}{t_j(q)} \leq \frac{q}{p} \quad \text{for all } 1 \leq p < q \leq p_j^{\max}. \tag{6}$$

Throughout this work, we assume that an optimal schedule allocates, for every task j, a number of processor $p_j \in [1, p_j^{\max}]$; using more than $p_j^{\max}$ processors would only lengthen execution and waste resources. Likewise, Fair never allocates more than $p_j^{\max}$ processors to any task.

Lemma 1. *All tasks following a speedup model introduced above are monotonic on $[1, p_j^{\max}]$.*

Proof. Because the Roofline, Communication, and Amdahl models are special cases of the General Model, it suffices to establish monotonicity for the General Model and for the Power-Communication Model. Once the communication overhead outweighs the parallelization gain (the term w_j/p_j), execution time can only increase with additional processors, while area cannot decrease. A detailed calculus-based proof is provided in Appendix A. □

3.3 Lower Bound on Optimal Makespan

In this section, we define the lower bound used to assess the quality of Fair via its ***competitive ratio***. To that end, we let t_j^{opt} and a_j^{opt} denote the execution time and area of task j under the processor allocation of an optimal schedule. We first define two quantities whose maximum will serve as the single lower bound used in our competitive-ratio analysis.

Definition 3. *Given the processor allocations of all the tasks in an optimal schedule,*

- *the* **total area** A^{opt} *of the task graph is the sum of all individual areas, i.e.,* $A^{\text{opt}} = \sum_{j=1}^{n} a_j^{\text{opt}}$
- *the length $L^{\text{opt}}(f)$ of a path[1] f in the graph is the sum of the execution times of all the tasks along that path, i.e., $L^{\text{opt}}(f) = \sum_{j \in f} t_j^{\text{opt}}$. The* **critical path length** C^{opt} *of the graph is the length of the longest path in the graph, i.e., $C^{\text{opt}} = \max_f L^{\text{opt}}(f)$.*

Clearly, the optimal makespan cannot be smaller than either $\frac{A^{\text{opt}}}{P}$ or C^{opt}. This follows from an adaptation from the well-known area and critical-path bounds for scheduling any task graph [11]. The only refinement is that a sharp analysis allows us to compare directly to an unknown optimal schedule, while most similar papers use a similar but looser bounds based on the instance, that can be computed in polynomial time. The following lemma states this result.

Lemma 2. $T^{\text{opt}} \geq \max\left(\frac{A^{\text{opt}}}{P}, C^{\text{opt}}\right)$.

3.4 Ratios and Main Result

We begin by defining a performance ratio for each task that balances execution time and processor usage.

Definition 4. *For a given task j, let $t_j(p)$ denote its execution time when allocated p processors, and let $a_j(p) = p \cdot t_j(p)$ denote its area. Define:*

- $t_j^{\min} \triangleq \min_{1 \leq p \leq P} t_j(p)$ *as the minimum execution time;*
- $a_j^{\min} \triangleq \min_{1 \leq p \leq P} a_j(p)$ *as the minimum area; by assumption, this minimum corresponds to $a_j(1)$;*
- *the ratio function* $R_j(p) \triangleq \max\left(\frac{t_j(p)}{t_j^{\min}}, \frac{a_j(p)}{a_j^{\min}}\right)$;
- *the allocation minimizing the ratio is $p_j = \arg\min_{1 \leq p \leq p_j^{\max}} R_j(p)$, where $p_j^{\max}$ minimizes $t_j(p)$.*

The final ratio for task j is then defined as $R_j \triangleq R_j(p_j)$.

Let $R = \max_j R_j$ be the global maximum over all tasks. This value captures the worst-case deviation from optimality across all tasks in terms of time and area and is used to define the competitive ratio of the new heuristic Fair.

Our main results are:

- The online scheduling algorithm Fair is $\frac{2R+1+\sqrt{4R^2+1}}{2} < 2R+1$-competitive: for any instance of moldable tasks with arbitrary speedup profiles, the resulting makespan is at most $(2R+1)$ times the optimal clairvoyant schedule.

[1]A path f consists of a sequence of tasks with linear dependencies, i.e., $f = (j_{\pi(1)}, \ldots, j_{\pi(v)})$, where the first task $j_{\pi(1)}$ has no predecessor, the last task $j_{\pi(v)}$ has no successor, and for each $2 \leq i \leq v$ task $j_{\pi(i)}$ is a successor of $j_{\pi(i-1)}$.

Algorithm 1: Compute_Ratio(j)

```
Input: Task $j$
                                                              // Task to be processed
Output: Ratio $R_j$ and processor allocation $p_j$
                                    // Optimal ratio and processor allocation for task $j$
// Compute initial allocation
1 Compute $p_j^{\max}$                                // Processor count minimizing execution time
2 Compute $t_j^{\min} = t_j(p_j^{\max})$ and $a_j^{\min} = a_j(1)$       // Minimum execution time and area
3 Find $p_j \in [1, p_j^{\max}]$ minimizing $R_j \triangleq \max\left(\frac{t_j(p)}{t_j^{\min}}, \frac{a_j(p)}{a_j^{\min}}\right)$      // Balance between time and area
4 return $R_j, p_j$
```

Algorithm 2: Fair

```
1  initialize waiting queue $Q$                                    // Queue of available tasks
2  $r = 1$                                                   // Initialize maximum ratio to 1
3  when at time 0 or a task completes do
       // Update task metrics
4      for each new task $j$ (that just became available/discovered) do
5          $(R_j, p_j) \leftarrow$ Compute_Ratio($j$)        // Compute ratio and allocation for task $j$
6          $r \leftarrow \max(r, R_j)$                           // Update maximum ratio if necessary
7          insert $j$ into $Q$ with $p_j$                            // Add task to waiting queue
8      end
       // List Scheduling
9      for each task $j$ in $Q$ do
10         $p'_j \leftarrow \min(p_j, \lceil \mu(r) \cdot P \rceil)$          // Scale processor allocation using $\mu(r)$
11         if available processors $\geq p'_j$ then
12             allocate $p'_j$ processors to $j$                   // Allocate processors to task $j$
13             execute $j$ immediately                           // Start execution of task $j$
14         end
15     end
16 end
```

- Moreover, Fair is $(2\sqrt{P}+3)$-competitive.
- Every online algorithm is at least $\left(\frac{\sqrt{P}-1}{2}\right)$-competitive. Hence, Fair is near-optimal in the worst case.

The rest of the paper defines our heuristic, proves these results, and then confirms the quality of Fair through experiments.

4 Fair Algorithm

In this section, we introduce Fair, an online scheduling algorithm for which we will derive a general competitive ratio in Section 5, along with specific results for the various speedup models previously discussed in Section 6. In contrast to earlier works [5, 21], where the algorithms were tailored to specific models, our approach provides a unified heuristic that applies to arbitrary tasks and achieves similar competitive guarantees for specific models.

We begin by describing the subroutine `Compute_Ratio`(j), which assigns an initial processor allocation to each task j and is detailed in Algorithm 1. For each possible allocation $p \in [1, p_j^{\max}]$, `Compute_Ratio`(j) computes $R_j(p)$ and finds the processor allocation minimizing it, noted p_j, with a resulting ratio R_j. Because this allocation minimizes the maximum of the two normalized metrics, it ensures that both the execution time and the area of task j are as close as possible of their respective optima. Since $g_j(p)$ is non-decreasing and $f_j(p)$ is non-increasing over p, this minimization can be efficiently solved using binary search in $O(\log P)$ time. The subroutine returns p_j and R_j.

Algorithm 2 presents the main Fair scheduling algorithm. The algorithm always maintains a queue Q of available (i.e., ready-to-execute) tasks. At time 0, and whenever a task finishes execution, it checks for newly available tasks. For each such task j, it invokes Algorithm 1 to compute its initial processor allocation and associated ratio, and inserts the task into Q.

If any of these new tasks result in a higher ratio than the current global ratio r, the algorithm updates r. This ensures that the current ratio reflects the maximal among all previously seen tasks, which is fundamental for the analysis. The scheduler then applies a standard ***list scheduling*** strategy [11]: it iterates over all tasks in Q and launches them immediately whenever enough processors are available. Note that no explicit priority order is imposed when inserting tasks into Q, although in practice, additional priority rules may further improve performance.

To improve processor utilization and avoid excessive idle time, the algorithm includes a reallocation step (line 10), that ensures that no task is assigned more than a certain fraction of the total processors. This mechanism is inspired by the processor capping strategy introduced in [20]. Although we lose the guarantee that each task's execution time remains within a factor r of the optimal execution time, it mitigates scenarios in which tasks would otherwise wait for a long time despite many idle processors, potentially leading to a large makespan. However, this cap shouldn't be too small; otherwise the resulting allocation could differ too significantly from the optimal.

In the analysis, we will show that the best choice for this fraction is denoted by:

$$\mu(r) = \frac{2r + 1 - \sqrt{4r^2 + 1}}{2r} \tag{7}$$

More precisely, let p_j denote the initial allocation for task j, and p'_j the final allocation after adjustment. Then

$$p'_j = \begin{cases} \lceil \mu(r)P \rceil, & \text{if } p_j > \lceil \mu(r)P \rceil \\ p_j, & \text{otherwise} \end{cases} . \tag{8}$$

Having now completed the description of Fair we turn to its performance. The next section develops a unified analysis framework that converts the per-task ratio bounds into a global makespan guarantee, yielding competitive-ratio results for each speed-up model introduced earlier.

5 General Analysis

We first outline a general analysis framework, under which the competitive ratio of the proposed online algorithm will be derived for different speedup models. The framework is inspired by the analysis shown in [20, 16, 15] for list scheduling as well as the analysis used in [5, 21] for local processor allocation, but the analysis is deeper and sharper. Together, the result nicely connects the makespan of the online algorithm to the lower bound (Lemma 2), thus proving the competitive ratio.

5.1 Preliminaries

We start with introducing the definitions required to carry out a tight analysis for the competitive ratio. These definitions will be separated in three parts, and an illustrating example will help the understanding at the end of each of the parts.

5.1.1 Intervals and Lengths

The schedule produced by the online scheduling algorithm Fair (Algorithm 2), can be partitioned into a set $\mathcal{I} = \{I_1, I_2, \dots\}$ of non-overlapping intervals, where tasks begin or end only at interval boundaries [2]. Consequently, within each interval $I \in \mathcal{I}$, the number of processors in use, denoted $p(I)$, and the current value of r, denoted $r(I)$, remain constant. Again, r represents the maximum value of $\max\left(\frac{t_j(p)}{t_j^{\min}}, \frac{a_j(p)}{a_j^{\min}}\right)$ observed among all tasks processed up to that point, updated solely upon task completions at interval endpoints (line 6 of Algorithm 2).

For each interval I, we define $\mu_I = \mu(R(I))$. The processor utilization $p(I)$ is the total number of processors allocated to tasks running during I. We classify the intervals in $\mathcal{I}$ into two primary categories based on processor utilization relative to the total number of processors P:

- $\mathcal{I}_0$: Intervals where $0 < p(I) < \lceil (1 - \mu_I)P \rceil$, representing periods of low to moderate processor usage.
- $\mathcal{I}_3$: Intervals where $\lceil (1 - \mu_I)P \rceil \le p(I) \le P$, indicating high processor utilization.

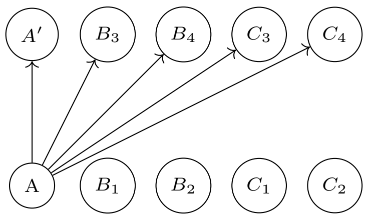
Clearly, $\mathcal{I}_0$ and $\mathcal{I}_3$ are well-defined and partition $\mathcal{I}$ (i.e., $\mathcal{I}_0 \cup \mathcal{I}_3 = \mathcal{I}$ and $\mathcal{I}_0 \cap \mathcal{I}_3 = \emptyset$). Indeed $p(I) > 0$ since Fair never leaves the platform idle. This classification highlights the distinction between underutilized and heavily utilized periods in the schedule.

A key property of $\mathcal{I}_0$ is established in the following lemma:

Lemma 3. *There exists a path f in the task graph such that at least one task from f is always running during intervals in $\mathcal{I}_0$.*

[2] Recall that all scheduling decisions are considered instantaneous in this work.

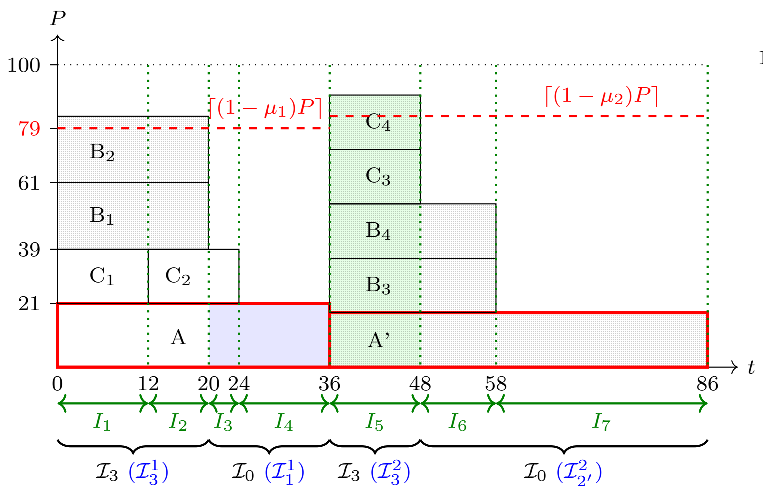
Proof. During intervals in $\mathcal{I}_0$, the processor utilization is at most $\lceil (1-\mu_I)P \rceil - 1$, which leaves at least $\lceil \mu_I P \rceil$ available processors[3]. Based on Algorithm 2, line 10, any task in the queue $\mathcal{Q}$ will be tried with at most $\lceil \mu_I P \rceil = \lceil \mu(R(I))P \rceil$ processors. Because any potential task in $\mathcal{Q}$ would fit, the queue has to be empty and there is no available task in the queue during $\mathcal{I}_0$. When a task graph is scheduled by a greedy list scheduling algorithm, such as FAIR, it is well known that there exists a path f in the graph such that whenever the queue is empty, some task on that path is running [10, 20, 16], hence the result. $\square$



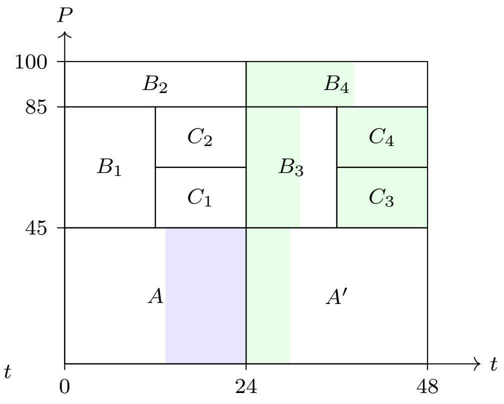
(a) Dependency graph

	A	A'	B_1	B_2	B_3	B_4	C_1	C_2	C_3	C_4
p	21	21	40				18			
R	2	2.6	2				1.5			
p'	21	18	22	22	18	18	18	18	18	18
t	36	50	20	20	22	22	12	12	13	13
$\boldsymbol{t}^{\text{opt}}$	24	24	12	24	12	24	12	12	12	12
$\boldsymbol{p}^{\text{opt}}$	45	45	40	15	40	15	20	20	20	20

(b) Tasks parameters



(c) FAIR schedule



(d) Optimal schedule

Figure 1: Overview of all introduced notations using an example. Top: (a) dependency graph and (b) summary table of execution time and processor allocation. Bottom: (c) FAIR schedule and (d) optimal schedule.

The intuition is as follows. The total time spent in $\mathcal{I}_0$ is limited, because there is always at least one task from a fixed path running during these intervals; hence that time cannot exceed the processing time of the path itself. Similarly, the time spent in $\mathcal{I}_3$ is bounded, since the algorithm processes the graph efficiently in that high-utilisation regime. By design, FAIR balances the allocation such that both execution time and area as are close as possible from the optima, hence these two facts provide the starting point for bounding the overall makespan.

Example To illustrate these concepts, we consider the example provided in Figure 1, which shows in (a) an instance of a task dependency graph. The task graph is composed of ten tasks structured in two layers: the first row contains the independent tasks A, B_1, B_2, C_1, and C_2, while the second row includes tasks A', B_3, B_4, C_3, and C_4, all of which depend on task A as a prerequisite. That is, none of the second-row tasks can be launched before task A has completed. All the B_i (resp. C_i) are identical tasks, hence the values (p,R) computed by COMPUTE_RATIO(j) are also identical.

The schedule produced by the FAIR algorithm for this instance is shown in (c), while an optimal schedule is shown in (d). The platform comprises $P = 100$ processors. According to the summary table in (b), the values of p_j and R_j for each task are determined by the function COMPUTE_RATIO(j). At time $t = 0$, the available tasks are A, B_1, B_2, C_1, and C_2, among which the maximum ratio is $R = 2$ (see the second row of the table), resulting in

[3]Indeed, if $x + y = P$ and P is integer, we can show $\lceil x \rceil + \lceil y \rceil \leq P + 1$ since the decimal part adds up to 0, if x is integer, or 1 otherwise since $x + y$ is integer.

a threshold of $\lceil \mu_1 P \rceil = 22$: no task can be allocated more than 22 processors. For this reason, B_1 and B_2 were **reduced** to 22 processors, and $p' \neq p$, which is represented in red in the table and visually indicated in the schedule in (c) by shaded boxes for the affected tasks.

Following the algorithm, tasks A, B_1, B_2, and C_1 are launched first. At this point, only 17 processors remain available, which correspond to a high processor allocation (above the red dotted $\lceil (1-\mu_1)P \rceil$ line), hence we are in $\mathcal{I}_0$. The amount of processor available is insufficient to launch C_2, which must stay in the queue until more processors are available at $t = 12$. From $t = 20$ until A completes at $t = 20$, the total processor usage falls below $(1-\mu_1)P$, directly implying that no tasks are available, otherwise one would have been launched by Fair.

At time $t = 36$, task A completes, releasing processors and making its dependent successors A', B_3, B_4, C_3, and C_4 available. Among these, the maximum ratio is now $R = 2.6$, as shown in the table, which updates the threshold to $\lceil \mu_2 P \rceil = 18$, so tasks may not be scheduled on more than 18 processors. Many of these tasks had larger original allocations (see row p in the table), and are thus reduced to respect the new threshold $\lceil \mu_2 P \rceil$, resulting in the adjusted allocations p'_j shown in red in the corresponding row.

Since enough processors are available at $t = 36$, all five newly available tasks can be launched simultaneously. Although the queue is empty, the processor usage is above $\lceil (1-\mu_2)P \rceil$. By $t = 48$, the processor usage again drops below $\lceil (1-\mu_2)P \rceil$ and the waiting queue is empty. The execution then continues until completion at time $t = 86$.

As described earlier, the schedule is partitioned into a sequence of intervals $I_1, I_2, \ldots$, each bounded by task start or end events. Within each interval, both the number of processors in use $p(I)$ and the value $\mu_I = \mu(R(I))$ remain constant. Intervals for which $p(I) \geq \lceil (1-\mu_I)P \rceil$ are classified in $\mathcal{I}_3$, representing high utilization phases, while those with $p(I) < \lceil (1-\mu_I)P \rceil$ fall into $\mathcal{I}_0$, corresponding to underutilization. As established in Lemma 3, during any interval in $\mathcal{I}_0$, the waiting queue is empty and there exists a path in the task graph such that one task from the path is always running. In this example, the path $A \to A'$ covers the two $\mathcal{I}_0$ intervals in the schedule.

Further Interval Subdivisions Next, we define a sequence of times $t_0 = 0 < t_1 < t_2 < \cdots < t_{m-1} < t_m = T$, where each t_k (for $k = 1, \ldots, m-1$) marks a task completion at which r is updated, and $t_m = T$ is the makespan. Since r updates occur only upon task completions and there are n tasks, we have $m \leq n$, a finite number. We subdivide the set of intervals $\mathcal{I}$ into m subsets defined by

$$\mathcal{I}^k = \{ I \in \mathcal{I} \mid I \subseteq [t_{k-1}, t_k] \}, \quad k = 1, 2, \ldots, m,$$

so that $(\mathcal{I}^k)_{k=1}^m$ forms a partition of $\mathcal{I}$.

We then define

$$\mathcal{I}_3^k = \mathcal{I}_3 \cap \mathcal{I}^k \quad \text{and} \quad \mathcal{I}_0^k = \mathcal{I}_0 \cap \mathcal{I}^k, \quad k = 1, 2, \ldots, m,$$

which denote the intervals of $\mathcal{I}_0$ and $\mathcal{I}_3$ that lie within the time period $[t_{k-1}, t_k]$, respectively. In each $\mathcal{I}_0^k$ and $\mathcal{I}_3^k$, the value $r(I)$ remains constant (denoted by r_k) and so does $\mu(r(I))$ (denoted by μ_k). We have $r_1 < r_2 < \cdots < r_m = R$, since it corresponds to the maximum ratio at the end of the schedule. Note that $\mathcal{I}_0$ and $\mathcal{I}_3$ partition $\mathcal{I}$, while the $(\mathcal{I}_0^k)_{k=1}^m$ and $(\mathcal{I}_3^k)_{k=1}^m$ partition $\mathcal{I}_0$ and $\mathcal{I}_3$, respectively.

Using the path f from Lemma 3, we further categorize each $\mathcal{I}_0^k$ into two subcategories based on the processor allocation decisions made by Fair:

- $\mathcal{I}_1^k$: Intervals in $\mathcal{I}_0^k$ where the task from f running retains its initial processor allocation p_j, as computed by `Compute_Ratio` (not reduced in line 10 of Algorithm 2).
- $\mathcal{I}_2^k$: Intervals in $\mathcal{I}_0^k$ where the task from f running has its processor allocation reduced to $\lceil \mu_k P \rceil$ processors (line 10 of Algorithm 2).

To compare with an optimal offline schedule, we split $\mathcal{I}_2^k$ further:

- $\mathcal{I}_{2'}^k$: Intervals in $\mathcal{I}_2^k$ where the task from f running is allocated strictly fewer processors than in the optimal schedule.
- $\mathcal{I}_{2''}^k$: Intervals in $\mathcal{I}_2^k$ where the task from f running is allocated at least as many processors as in the optimal schedule.

To provide a clearer understanding of the impact of each interval type on the performance of the Fair algorithm, we now give an intuitive explanation of their behavior:

- In intervals $\mathcal{I}_1^k$, the task from path f is allocated the full processor count p_j as selected by `Compute_Ratio`, without any reduction. By construction, this allocation guarantees that both the execution time and the area are within a factor r_k of their respective minima. Consequently, the task progresses along path f at most r_k times more slowly than in the optimal schedule.

- In intervals $\mathcal{I}^k_{2'}$, the task from f is reduced to $\lceil \mu_k P \rceil$ processors, which is fewer than its allocation in the optimal schedule. As we assume no superlinear speedup, the execution of the task in f progresses at most μ_k times slower than in the optimal (assuming in the worst case scenario that the optimal uses all P processors with perfect speedup). Moreover, the area progresses at most r_k/μ_k times more slowly than in the optimal schedule, because the allocation for this task given by `Compute_Ratio` remains within a factor r_k of its minimum area, and the allocation is reduced compared to its original value, which may only decrease the area. Since at least $\mu_k P$ processors are used, no schedule may use more than $\frac{1}{\mu_k}$ times more processors, while being at most r_k times more efficient with the processors.
- In intervals $\mathcal{I}^k_{2''}$, the task from f is again reduced to $\lceil \mu_k P \rceil$ processors, but this time receives at least as many processors as in the optimal schedule. Thus, progress along the path is guaranteed to be no worse than that in the optimal using the monotonicity assumption, and the efficiency for the area is again no worse than a factor r_k/μ_k from the optimal.
- Finally, in intervals $\mathcal{I}^k_3$, at least $(1-\mu_k)P$ processors are active. Since all running tasks have been allocated in a way that ensures an area within a factor r_k of the minimum, the aggregate area consumption is at most $r_k/(1-\mu_k)$ times slower than the corresponding area in the optimal schedule.

This refined classification will allow the derivation of bounds involving $\frac{A^{\text{opt}}}{P}$ and C^{opt} , while accounting for the varying values of r_k that arise dynamically in the execution of Fair. This analysis builds upon the methodology introduced in [21], but extends it to accommodate the variation of the reduction step in Line 10 of Algorithm 2, which was not addressed in earlier work.

For each interval I, let $|I|$ denote its duration. We define the total durations of each category within period k:

$$T^k_1 = \sum_{I\in\mathcal{I}^k_1} |I|, \quad T^k_2 = \sum_{I\in\mathcal{I}^k_2} |I|, \quad T^k_{2'} = \sum_{I\in\mathcal{I}^k_{2'}} |I|, \quad T^k_{2''} = \sum_{I\in\mathcal{I}^k_{2''}} |I|, \quad T^k_3 = \sum_{I\in\mathcal{I}^k_3} |I|.$$

Given the partitioning properties—$\mathcal{I}_0$ and $\mathcal{I}_3$ partition $\mathcal{I}$, $(\mathcal{I}^k_0)^m_{k=1}$ partition $\mathcal{I}_0$, $(\mathcal{I}^k_3)^m_{k=1}$ partition $\mathcal{I}_3$, $\mathcal{I}^k_1$ and $\mathcal{I}^k_2$ partition $\mathcal{I}^k_0$, and $\mathcal{I}^k_{2'}$ and $\mathcal{I}^k_{2''}$ partition $\mathcal{I}^k_2$—the makespan is:

$$T = \sum_{k=1}^{m} (T^k_1 + T^k_2 + T^k_3).$$

This structured decomposition enables a detailed worst-case performance analysis of Fair by examining the contributions of each interval category to the makespan T.

In the rest of the analysis, we will build upper bounds that will be used to replace the T^k_1, T^k_2 and T^k_3 to eventually show that $T \leq \frac{T_{opt}}{\mu(R)}$, where R corresponds to the largest ratio among all tasks.

Additionally, for each k, since $T^k_2 = T^k_{2'} + T^k_{2''}$, we introduce $z_k \in [0,1]$ such that:

$$T^k_{2'} = z_k T^k_2, \quad T^k_{2''} = (1-z_k) T^k_2 \tag{9}$$

quantifying the proportion of $\mathcal{I}^k_2$ where the algorithm under-allocates processors for tasks in the path f compared to the optimal and vice versa.

Finally, for each period k, we define the total time task j runs within $\mathcal{I}^k$ in the Fair schedule:

$$T^k_j = \sum_{I\in\mathcal{I}^k} |I \cap [s_j, e_j]|$$

. Similarly, we can subdivide this length based on the partition of $\mathcal{I}^k$:

$$T^k_{j,1} = \sum_{I\in\mathcal{I}^k_1} |I \cap [s_j, e_j]| \quad T^k_{j,2'} = \sum_{I\in\mathcal{I}^k_{2'}} |I \cap [s_j, e_j]|, \quad T^k_{j,2''} = \sum_{I\in\mathcal{I}^k_{2''}} |I \cap [s_j, e_j]|, \quad T^k_{j,3} = \sum_{I\in\mathcal{I}^k_3} |I \cap [s_j, e_j]|.$$

For example, $T^k_{j,2'}$ corresponds to the total amount of time where task j runs within the intervals in $\mathcal{I}^k_{2'}$.

Example In the example of Figure 1, there is a single update to the value of r during the schedule: it increases from $r_1 = 2$ to $r_2 = 2.6$ when task A' becomes available at time $t = 36$. This creates a subdivision of the time axis into two intervals: $[0, 36]$ and $[36, 86]$, corresponding to $k = 1$ and $k = 2$, respectively, so $m = 2$.

The first high-utilization phase, I_3 from $[0, 20]$ (highlighted in blue in the figure), belongs to $\mathcal{I}_3^1$. The following interval $[20, 36]$ falls into $\mathcal{I}_1^1$, and since task A—the task from path $f = A \to A'$ running during this interval—retains its original allocation (i.e., it is not reduced), this interval belongs to $\mathcal{I}_1^1$.

The second high-utilization phase, from $[36, 48]$, is part of $\mathcal{I}_3^2$. Finally, the last phase from $[48, 86]$ belongs to $\mathcal{I}_0^2$. During this period, task A' from path f is running with a reduced processor allocation $p' = 18$, which is strictly less than the $p^{\text{opt}} = 45$ processors used in the optimal schedule (see table (b) and schedule (d)). Therefore, this interval belongs to $\mathcal{I}_{2'}^2$.

In this simple example, each category contains only a single interval, so the corresponding durations T_3^1, T_1^1, T_3^2 and $T_{2'}^2$, can be directly read as the lengths of those intervals: 20, 16, 12 and 38, respectively. In general, these totals are obtained by summing the lengths of all intervals in each category, even when they are non-contiguous. Any other interval type, such as $I_{2'}^1$ is not represented in this example and $T_{2'}^1 = 0$.

Let us now consider the execution time of task A within these subdivisions. Task A starts at $t = 0$ and completes at $t = 36$, intersecting both the interval in $\mathcal{I}_3^1$ and the one in $\mathcal{I}_1^1$. The duration of A overlapping with the first high-utilization interval (I_3^1) is $|[0, 20]| = 20$, which gives $T_{A,3}^1 = 20$. Similarly, the duration overlapping with the underutilized interval ($\mathcal{I}_1^1$) is $|[20, 36]| = 16$, giving $T_{A,1}^1 = 16$. For other interval categories, either the one that appears in the schedule where A doesn't intersect, such as $\mathcal{I}_3^2$; or the ones that don't appear, the corresponding time is 0. For instance, $T_{A,2''}^1 = T_{A,3}^2 = 0$.

This breakdown highlights how each interval contributes to the makespan and to the execution timeline of individual tasks in the path f. Such fine-grained decomposition is essential to derive performance bounds for the Fair algorithm.

5.1.2 Areas

To analyze the worst-case performance of the Fair algorithm, we introduce area-based metrics that quantify the resource utilization of tasks during specific intervals within each period k. These metrics enable a comparison between the Fair schedule and an optimal schedule, supporting the derivation of upper bounds on the makespan T.

Areas in the Fair Schedule The areas associated with tasks in the path f in the Fair schedule are defined as

$$A_{2'|f}^k = \sum_{j \in f} p_j' \, T_{j,2'}^k, \quad A_{2''|f}^k = \sum_{j \in f} p_j' \, T_{j,2''}^k, \quad A_3^k = \sum_{j=1}^{n} p_j' \, T_{j,3}^k,$$

where p_j' denotes the number of processors allocated to task j in the Fair schedule. Note that we only include tasks in the path f for $A_{2'|f}^k$ and $A_{2''|f}^k$. The reason is that, in the worst case, no other task may be running during intervals of type $\mathcal{I}_{2'}^k$ or $\mathcal{I}_{2''}^k$; hence including additional tasks would not tighten the competitive-ratio bound. Conversely, all tasks are taken into account in A_3^k. These definitions capture the resource utilization of tasks during the subintervals of period k in the Fair schedule.

Areas in the Optimal Schedule To facilitate comparison, we define corresponding areas in the optimal schedule based on the proportional execution times of tasks in the Fair schedule. In particular, we define

$$A_{2'|f}^{k,\text{opt}} = \sum_{j \in f} \frac{T_{j,2'}^k}{t_j(p_j')} a_j^{\text{opt}}, \quad A_{2''|f}^{k,\text{opt}} = \sum_{j \in f} \frac{T_{j,2''}^k}{t_j(p_j')} a_j^{\text{opt}}, \quad A_3^{k,\text{opt}} = \sum_{j=1}^{n} \frac{T_{j,3}^k}{t_j(p_j')} a_j^{\text{opt}}, \quad A^{k,\text{opt}} = \sum_{j=1}^{n} \frac{T_j^k}{t_j(p_j')} a_j^{\text{opt}}$$

where $a_j^{\text{opt}} = p_j^{\text{opt}} t_j(p_j^{\text{opt}})$ is the area of task j in the optimal schedule, p_j^{opt} denotes the optimal processor allocation, and $t_j(p_j')$ is the execution time of task j with p_j' processors in the Fair schedule. These definitions capture the proportional areas in the optimal schedule corresponding to the time spent in the intervals $\mathcal{I}_{2'}^k$, $\mathcal{I}_{2''}^k$, $\mathcal{I}_3^k$, and $\mathcal{I}^k$ respectively, in the Fair schedule.

Example In the example of Figure 1, (c) and (d), we illustrate in green the area calculations for A_3^2 in the Fair schedule (left), and $A_3^{2,\text{opt}}$ in the optimal schedule (right). These quantities correspond to the total processor-time product (i.e., area) used by all tasks during the high-utilization interval $I_3^2 = [36, 48]$.

To compute A_3^2, we consider all tasks that are running during I_3^2 in the Fair schedule. This includes tasks C_3 and C_4, which are fully contained in this interval and use 18 processors each. In addition, tasks B_3 and B_4 overlap partially with I_3^2: they run for 22 time units total, and only the first 12 time units lie within I_3^2. Therefore, each of them contributes $\frac{12}{22}$ of its area. Finally, task A' overlaps I_3^2 for its first 12 units of execution out of 50 total, contributing $\frac{12}{50}$ of its area. The total area in FAIR is thus:

$$A_3^2 = a_{C_3} + a_{C_4} + \frac{12}{22}(a_{B_3} + a_{B_4}) + \frac{12}{50}a_{A'},$$

where $a_j = p'_j \cdot t_j(p'_j)$ denotes the area of task j in the FAIR schedule.

To compute the corresponding area in the optimal schedule, we use the same fractions of execution but apply them to the optimal areas. These tasks are highlighted in green in the optimal schedule (d). The total area is given by:

$$A_3^{2,\text{opt}} = a_{C_3}^{\text{opt}} + a_{C_4}^{\text{opt}} + \frac{12}{22}(a_{B_3}^{\text{opt}} + a_{B_4}^{\text{opt}}) + \frac{12}{50}a_{A'}^{\text{opt}},$$

where $a_j^{\text{opt}} = p_j^{\text{opt}} \cdot t_j(p_j^{\text{opt}})$ is the optimal area of task j. This definition reflects the idea that the area in the optimal schedule is reconstructed by proportionally allocating task areas according to the fraction of time those tasks run in the FAIR schedule during I_3^2.

This proportional view enables us to compare processor usage efficiency between FAIR and an optimal schedule, interval by interval, and ultimately derive bounds on the total makespan.

5.1.3 Paths

Recall that $T_{j,1}^k$ corresponds to the total length of time that task j runs within the intervals $\mathcal{I}_1^k$ in the FAIR schedule (resp. $T_{j,2'}^k$, $T_{j,2''}^k$). We define the corresponding lengths in the optimal schedule by scaling these durations according to the fraction of task j executed in the FAIR schedule, multiplied by the optimal execution time t_j^{opt} for task j:

$$L_{1|f}^{k,\text{opt}} = \sum_{j\in f} \frac{T_{j,1}^k}{t_j(p'_j)} t_j^{\text{opt}}, \quad L_{2'|f}^{k,\text{opt}} = \sum_{j\in f} \frac{T_{j,2'}^k}{t_j(p'_j)} t_j^{\text{opt}}, \quad L_{2''|f}^{k,\text{opt}} = \sum_{j\in f} \frac{T_{j,2''}^k}{t_j(p'_j)} t_j^{\text{opt}}, \quad L_{f}^{k,\text{opt}} = \sum_{j\in f} \frac{T_{j}^k}{t_j(p'_j)} t_j^{\text{opt}},$$

where $t_j(p'_j)$ denotes the execution time of task j in the FAIR schedule and t_j^{opt} its execution time under an optimal schedule. These definition follows exactly the same idea as the area definitions.

Example In the example of Figure 1, (c) and (d), we highlight in blue on the left the portion of the path $f = A \to A'$ that overlaps the interval $I_1^1 = [20, 36]$, which is part of $\mathcal{I}_1^1$. During this interval, only task A is running on the path f, and it overlaps with the full 16 units of this interval out of its total execution time of 36 in the FAIR schedule. Therefore, the proportion of task A that intersects $\mathcal{I}_1^1$ is $\frac{16}{36}$.

To compute the corresponding optimal length, we use this fraction scaled by the optimal execution time of A, yielding:

$$L_{1|f}^{1,\text{opt}} = \frac{16}{36} \cdot t_A^{\text{opt}}.$$

From the task table (b), we know that $t_A^{\text{opt}} = 24$, so this gives $L_{1|f}^{1,\text{opt}} = \frac{16}{36} \cdot 24$. In this example, A is the only task in f that overlaps $\mathcal{I}_1^1$, but in general, several tasks along the path may contribute to the sum.

This concludes our concrete example-based illustration of the area and path metrics. All the ingredients are now in place to derive the approximation ratio of the FAIR algorithm by bounding the total makespan T in terms of the optimal makespan T^{opt}. But first, we will list and prove different properties that link the attributes introduced here.

5.2 Structural Properties of the Schedule

Table 3 collects several useful properties that are repeatedly used throughout the analysis. By grouping them in one place, we streamline the presentation of more involved derivations and avoid interrupting the main proofs with technical but straightforward steps. All properties listed in the table will be formally proved in this section.

Global Recall that the function μ is defined for $x \geq 1$ as $\mu(x) = \frac{2x+1-\sqrt{4x^2+1}}{2x}$. Hence, $\mu(x)$ has the same sign as $2x + 1 - \sqrt{4x^2 + 1} > 2x + 1 - \sqrt{4x^2 + 4x + 1} = 0$, hence it is well defined and positive. To show the Global results,

Table 3: Crucial properties

Global	Area	Path
$\frac{x}{1-\mu(x)} = \frac{1}{\mu(x)} - x$ (G1)	$\mu_k T^k_{2'} \leq \frac{A^{k,\text{opt}}_{2'\|f}}{P}$ (A1)	$T^k_1 \leq R_k L^{k,\text{opt}}_{1\|f}$ (P1)
$\frac{x}{1-\mu(x)}$ increases with x (G2)	$\frac{\mu_k T^k_{2''}}{R_k} \leq \frac{A^{k,\text{opt}}_{2''\|f}}{P}$ (A2)	$T^k_{2'} \leq \frac{L^{k,\text{opt}}_{2'\|f}}{\mu_k}$ (P2)
$\mu(x)$ decreases with x (G3)	$\frac{(1-\mu_k)T^k_3}{R_k} \leq \frac{A^{k,\text{opt}}_3}{P}$ (A3)	$T^k_{2''} \leq L^{k,\text{opt}}_{2''\|f}$ (P3)
$\mu_k \leq \frac{3-\sqrt{5}}{2}$ (G4)	$A^{k,\text{opt}}_{2'\|f} + A^{k,\text{opt}}_{2''\|f} + A^{k,\text{opt}}_3 \leq A^{k,\text{opt}}$ (A4)	$L^{k,\text{opt}}_{1\|f} + L^{k,\text{opt}}_{2'} + L^{k,\text{opt}}_{2''} \leq L^{k,\text{opt}}_f$ (P4)
$T^k_2 = \frac{T^k_{2'}}{z_k} = \frac{T^k_{2''}}{1-z_k}$ (G5)	$\sum_{k=1}^m A^{k,\text{opt}} \leq T^{\text{opt}}$ (A5)	$\sum_{k=1}^m L^{k,\text{opt}}_f \leq T^{\text{opt}}$ (P5)

we first define $v(x) = \frac{x}{1-\mu(x)}$

$$\begin{aligned} v(x) &= \frac{x}{1-\mu(x)} = \frac{x}{\frac{\sqrt{4x^2+1}-1}{2x}} = \frac{2x^2}{\sqrt{4x^2+1}-1} = \frac{2x^2(\sqrt{4x^2+1}+1)}{4x^2+1-1} = \frac{\sqrt{4x^2+1}+1}{2} \\ &= \frac{2x+1+\sqrt{4x^2+1}}{2} - x = \frac{(2x+1+\sqrt{4x^2+1})(2x+1+\sqrt{4x^2+1})}{2(2x+1-\sqrt{4x^2+1})} - x \\ &= \frac{(2x+1)^2-(4x^2+1)}{2(2x+1-\sqrt{4x^2+1})} - x = \frac{2x}{2x+1-\sqrt{4x^2+1}} - x = \frac{1}{\mu(x)} - x. \quad (\text{G1} \checkmark) \end{aligned}$$

Since $v(x) = \frac{\sqrt{4x^2+1}+1}{2}$ (first line, last equality), we see that $v(x)$ is increasing (G2 ✓), and with $v(x) = \frac{1}{\mu(x)} - x$, we have shown $\frac{1}{\mu(x)}$ must increase since $v(x)$ increases and $-x$ decreases with x. Thus $\mu(x)$ decreases with x (G3 ✓). Finally since the ratio is always larger than 1, $\mu(x) \leq \mu(1) = \frac{3-\sqrt{5}}{2}$ (G4 ✓). Using Equation 9, we directly have G5 ✓: it is simply the definition of z_k restated here for convenience.

Area For convenience we reproduce the area definitions: $A^k_{2'|f} = \sum_{j\in f} p'_j T^k_{j,2'}$, $A^k_{2''|f} = \sum_{j\in f} p'_j T^k_{j,2''}$, $A^k_3 = \sum_{j=1}^n p'_j T^k_{j,3}$, and $A^{k,\text{opt}}_{2'|f} = \sum_{j\in f} \frac{T^k_{j,2'}}{t_j(p'_j)} a^{\text{opt}}_j$, $A^{k,\text{opt}}_{2''|f} = \sum_{j\in f} \frac{T^k_{j,2''}}{t_j(p'_j)} a^{\text{opt}}_j$, $A^{k,\text{opt}}_3 = \sum_{j=1}^n \frac{T^k_{j,3}}{t_j(p'_j)} a^{\text{opt}}_j$, $A^{k,\text{opt}} = \sum_{j=1}^n \frac{T^k_j}{t_j(p'_j)} a^{\text{opt}}_j$.

(A1) In subintervals of $\mathcal{I}^k_{2'}$, tasks from path f utilize at least $\mu_k P$ processors (by definition). Thus, we have:

$$\mu_k P\, T^k_{2'} \ \leq \ A^k_{2'|f} = \sum_{j\in f} p'_j T^k_{j,2'} = \sum_{j\in f} \frac{a_j(p'_j)}{t_j(p'_j)} T^k_{j,2'} \leq \sum_{j\in f} \frac{a_j(p^{\text{opt}}_j)}{t_j(p'_j)} T^k_{j,2'} = A^{k,\text{opt}}_{2'|f},$$

where the inequality follows from the fact that area is non-decreasing with processor allocation: by definition, in $\mathcal{I}^k_{2'}$, all tasks j from path f are allocated fewer processors than in the optimal, hence $a_j(p'_j) \leq a_j(p^{\text{opt}}_j)$. Dividing by P, we obtain:

$$\mu_k T^k_{2'} \ \leq \ \frac{A^{k,\text{opt}}_{2'|f}}{P} \quad (\text{A1} \checkmark).$$

(A2) Again, in subintervals of $\mathcal{I}^k_{2''}$, tasks from f occupy at least $\mu_k P$ processors. Then:

$$\mu_k P\, T^k_{2''} \leq A^k_{2''|f} = \sum_{j\in f} p'_j T^k_{j,2''} = \sum_{j\in f} \frac{a_j(p'_j)}{t_j(p'_j)} T^k_{j,2''} \leq \sum_{j\in f} \frac{r_k a^{\text{opt}}_j}{t_j(p'_j)} T^k_{j,2''} = r_k A^{k,\text{opt}}_{2''|f},$$

This time, using the bound $a_j(p'_j) \leq a_j(p_j) \leq r_k a^{\min}_j \leq r_k a^{\text{opt}}_j$. All tasks in f processed in $\mathcal{I}^k_{2''}$ are a factor r_k away from the minimal area before reduction, hence it is still the case after reduction. Dividing by $r_k P$ yields:

$$\frac{\mu_k T^k_{2''}}{r_k} \leq \frac{A^{k,\text{opt}}_{2''|f}}{P} \quad (\text{A2} \checkmark).$$

(A3) In subintervals $\mathcal{I}_3^k$, at least $(1-\mu_k)P$ processors are used. We obtain:

$$(1-\mu_k)PT_3^k \le A_3^k = \sum_{j=1}^{n} p'_j T_{j,3}^k = \sum_{j=1}^{n} \frac{a_j(p'_j)}{t_j(p'_j)} T_{j,3}^k \le \sum_{j=1}^{n} \frac{r_k a_j^{\mathrm{opt}}}{t_j(p'_j)} T_{j,3}^k = r_k A_3^{k,\mathrm{opt}}.$$

For the same reason as in (A2), the bound $a_j(p'_j) \le r_k a_j^{\mathrm{opt}}$ holds. Dividing by $r_k P$ gives:

$$\frac{(1-\mu_k)T_3^k}{r_k} \le \frac{A_3^{k,\mathrm{opt}}}{P} \quad (\text{A3} \checkmark).$$

(A4) The terms $A_{2'|f}^{k,\mathrm{opt}}$, $A_{2''|f}^{k,\mathrm{opt}}$, and $A_3^{k,\mathrm{opt}}$ correspond to the proportional areas in the optimal schedule corresponding to the time spent in the intervals $\mathcal{I}_{2'}^k$, $\mathcal{I}_{2''}^k$, $\mathcal{I}_3^k$, and $\mathcal{I}^k$ respectively, in the Fair schedule. Since $\mathcal{I}_{2'}^k$, $\mathcal{I}_{2''}^k$, $\mathcal{I}_3^k$ are disjoint subsets of $\mathcal{I}^k$, all corresponding areas in the optimal schedule also correspond to disjoint fractions of the tasks (see the example in Figure 1, (c) and (d)). More precisely, $T_{j,2'}^k + T_{j,2''}^k + T_{j,3}^k \le T_j^k$. Hence,

$$A_{2'|f}^{k,\mathrm{opt}} + A_{2''|f}^{k,\mathrm{opt}} + A_3^{k,\mathrm{opt}} \le \sum_{j\in f} \frac{T_{j,2'}^k}{t_j(p'_j)} a_j^{\mathrm{opt}} + \sum_{j\in f} \frac{T_{j,2''}^k}{t_j(p'_j)} a_j^{\mathrm{opt}} + \sum_{j=1}^{n} \frac{T_{j,3}^k}{t_j(p'_j)} a_j^{\mathrm{opt}} \le \sum_{j=1}^{n} \frac{T_j^k}{t_j(p'_j)} a_j^{\mathrm{opt}} \le A^{k,\mathrm{opt}} \quad (\text{A4} \checkmark)$$

(A5) Finally, summing over all periods, we have:

$$\sum_{k=1}^{m} A^{k,\mathrm{opt}} = \sum_{k=1}^{m}\sum_{j=1}^{n} \frac{T_j^k}{t_j(p'_j)} a_j^{\mathrm{opt}}.$$

Now, for each task j, the total time it is executed over all intervals satisfies $\sum_{k=1}^{m} T_j^k = t_j(p'_j)$. Hence:

$$\sum_{k=1}^{m} A^{k,\mathrm{opt}} = \sum_{j=1}^{n} \frac{a_j^{\mathrm{opt}}}{t_j(p'_j)} \sum_{k=1}^{m} T_j^k = \sum_{j=1}^{n} a_j^{\mathrm{opt}} = A^{\mathrm{opt}} \quad (\text{A5} \checkmark).$$

Paths For convenience we reproduce the path definitions: $L_{1|f}^{k,\mathrm{opt}} = \sum_{j\in f} \frac{T_{j,1}^k}{t_j(p'_j)} t_j^{\mathrm{opt}}$, $L_{2'|f}^{k,\mathrm{opt}} = \sum_{j\in f} \frac{T_{j,2'}^k}{t_j(p'_j)} t_j^{\mathrm{opt}}$, $L_{2''|f}^{k,\mathrm{opt}} = \sum_{j\in f} \frac{T_{j,2''}^k}{t_j(p'_j)} t_j^{\mathrm{opt}}$, $L_f^{k,\mathrm{opt}} = \sum_{j\in f} \frac{T_j^k}{t_j(p'_j)} t_j^{\mathrm{opt}}$,

(P1) In subintervals in $\mathcal{I}_1^k$, the schedule executes exactly one task from the path f at all times (by Lemma 3, because $\mathcal{I}_1^k \subset \mathcal{I}_0$). Therefore, the total time spent in $\mathcal{I}_1^k$ satisfies:

$$T_1^k = \sum_{j\in f} T_{j,1}^k,$$

Again, this is because $T_{j,1}^k$ represents the amount of time spent in $\mathcal{I}_1^k$ by task j. No reduction is applied to tasks from f in $\mathcal{I}_1^k$, so by the definition of r_k, we have:

$$t_j(p'_j) \le r_k\, t_j^{\mathrm{opt}},$$

and hence:

$$T_1^k = \sum_{j\in f} \frac{T_{j,1}^k}{t_j(p'_j)} t_j(p'_j) \le r_k \sum_{j\in f} \frac{T_{j,1}^k}{t_j(p'_j)} t_j^{\mathrm{opt}} = r_k L_{1|f}^{k,\mathrm{opt}} \quad (\text{P1} \checkmark).$$

(P2) In subintervals in $\mathcal{I}_{2'}^k$, exactly one task from f is executed at all times. This task is reduced and uses $\lceil \mu_k P \rceil$ processors. Since we assume there is no superlinear speedup, and at most P processors are available, for all task j running in $\mathcal{I}_{2'}^k$,

$$t_j(p'_j) \le \frac{t_j^{\mathrm{opt}}}{\mu_k},$$

Therefore:

$$T_{2'}^k = \sum_{j\in f} T_{j,2'}^k = \sum_{j\in f} \frac{T_{j,2'}^k}{t_j(p'_j)} t_j(p'_j) \le \sum_{j\in f} \frac{T_{j,2'}^k}{t_j(p'_j)} \cdot \frac{t_j^{\mathrm{opt}}}{\mu_k} = \frac{1}{\mu_k} \sum_{j\in f} \frac{T_{j,2'}^k}{t_j(p'_j)} t_j^{\mathrm{opt}} = \frac{1}{\mu_k} L_{2'|f}^{k,\mathrm{opt}} \quad (\text{P2} \checkmark).$$

(P3) In subintervals in $\mathcal{I}_{2''}^k$, again one task from f runs at all times, but this time with at least as many processors as in the optimal schedule. Since using more processors can only reduce execution time due to the non-increasing nature of $t_j(p)$ on $[1, p_j^{\max}]$, we have:

$$t_j(p'_j) \le t_j^{\text{opt}} \Rightarrow T_{2''}^k = \sum_{j \in f} T_{j,2''}^k \le \sum_{j \in f} \frac{T_{j,2''}^k}{t_j(p'_j)} t_j^{\text{opt}} = L_{2''|f}^{k,\text{opt}} \quad (\text{P3} \checkmark).$$

(P4) Since the intervals in $\mathcal{I}_1^k$, $\mathcal{I}_{2'}^k$, and $\mathcal{I}_{2''}^k$ are disjoint, and exactly one task from path f is always running in these intervals. The corresponding optimal lengths can therefore be added:

$$L_{1|f}^{k,\text{opt}} + L_{2'|f}^{k,\text{opt}} + L_{2''|f}^{k,\text{opt}} \le L_f^{k,\text{opt}} \quad (\text{P4} \checkmark).$$

(P5) Finally, summing over all k:

$$\sum_{k=1}^m L_f^{k,\text{opt}} = \sum_{k=1}^m \sum_{j \in f} \frac{T_j^k}{t_j(p'_j)} t_j^{\text{opt}}.$$

For each task $j \in f$, the total time executed over all intervals satisfies $\sum_{k=1}^m T_j^k = t_j(p'_j)$, hence:

$$\sum_{k=1}^m L_f^{k,\text{opt}} = \sum_{j \in f} \frac{t_j^{\text{opt}}}{t_j(p'_j)} \sum_{k=1}^m T_j^k = \sum_{j \in f} t_j^{\text{opt}} = L_f^{\text{opt}}.$$

Since the critical path f is a subset of all tasks and the optimal makespan must be at least the length of any path:

$$\sum_{k=1}^m L_f^{k,\text{opt}} \le T^{\text{opt}} \quad (\text{P5} \checkmark).$$

In the next section, we combine all the results from Table 3 with the identity $T = \sum_{k=1}^m (T_1^k + T_2^k + T_3^k)$ to derive an upper bound on T in terms of T^{opt}.

5.3 Final Ratio Bound

Before stating the two key lemmas, we pause for a brief recap: many symbols have been introduced, and it is useful to gather their meaning in one place.

Time-partition During each window $[t_{k-1}, t_k]$ of the schedule produced by Fair, the intervals are split into disjoint sets: $\mathcal{I}_1^k$ (under-utilised, *unreduced* allocations on the path f), $\mathcal{I}_2^k$ (under-utilised, *reduced* allocations, meaning that at least $\lceil \mu_k P \rceil$ processors are used), and $\mathcal{I}_3^k$ (high utilisation). Their total lengths are T_1^k, T_2^k, and T_3^k, respectively. Inside $\mathcal{I}_2^k$ we further isolate a fraction z_k where the path task receives fewer processors than in the optimum ($\mathcal{I}_{2'}^k$) and a fraction $1 - z_k$ where it receives at least as many ($\mathcal{I}_{2''}^k$).

Efficiency parameters. $\mu_k = \mu(r_k)$ is the capping ratio that limits any processor allocation to $\mu_k P$ during period k, while r_k (≥ 1) is the largest per-task slowdown encountered so far, both for time and area. These two quantities measure how far Fair can deviate from an optimal allocation for these metrics.

Optimal benchmarks. $A^{k,\text{opt}}$ is the amount of area in the optimal offline schedule that corresponds to the fraction of tasks that Fair executes inside $[t_{k-1}, t_k]$. Likewise, $L_f^{k,\text{opt}}$ is the time the optimal schedule spends on the portion of the critical path f corresponding to the work done by Fair in that window. Together, they allow building bounds with the optimal time of the schedule.

Splitting time into $\mathcal{I}_1^k$, $\mathcal{I}_2^k$, and $\mathcal{I}_3^k$ provides detailed insights into Fair 's efficiency: $\mathcal{I}_1^k$ and $\mathcal{I}_2^k$ determine how quickly progress is made along the critical path, while $\mathcal{I}_2^k$ and $\mathcal{I}_3^k$ reflect how efficiently area is consumed when the platform is highly loaded. Each of the small properties derived in the previous subsection is a formalization of one of these ideas, and they will now be combined to derive the competitive ratio. The following two lemmas convert these local properties into global bounds by relating (T_1^k, T_2^k, T_3^k) to the optimal references $(A^{k,\text{opt}}, L_f^{k,\text{opt}})$. Each lemma corresponds to one of these bounds.

Lemma 4. *For any $k \in [1, m]$, we have:*

$$\mu_k \left(z_k + \frac{1 - z_k}{r_k} \right) T_2^k + \frac{1 - \mu_k}{r_k} T_3^k \le \frac{A^{k,\text{opt}}}{P} \ . \tag{10}$$

Proof. From property (G5) in Table 3, we have:

$$T_2^k = \frac{T_{2'}^k}{z_k} = \frac{T_{2''}^k}{1-z_k} \quad \Rightarrow \quad T_{2'}^k = z_k T_2^k, \quad T_{2''}^k = (1-z_k)T_2^k.$$

Using properties (A1), (A2), and (A3), we obtain:

$$\mu_k T_{2'}^k \le \frac{A_{2'|f}^{k,\text{opt}}}{P}, \qquad \frac{\mu_k T_{2''}^k}{r_k} \le \frac{A_{2''|f}^{k,\text{opt}}}{P}, \qquad \frac{(1-\mu_k)T_3^k}{r_k} \le \frac{A_3^{k,\text{opt}}}{P}.$$

Adding these inequalities:

$$\mu_k T_{2'}^k + \frac{\mu_k T_{2''}^k}{r_k} + \frac{(1-\mu_k)T_3^k}{r_k} \le \frac{A_{2'|f}^{k,\text{opt}} + A_{2''|f}^{k,\text{opt}} + A_3^{k,\text{opt}}}{P}.$$

Then, applying property (A4):

$$A_{2'|f}^{k,\text{opt}} + A_{2''|f}^{k,\text{opt}} + A_3^{k,\text{opt}} \le A^{k,\text{opt}},$$

we get:

$$\mu_k T_{2'}^k + \frac{\mu_k T_{2''}^k}{r_k} + \frac{(1-\mu_k)T_3^k}{r_k} \le \frac{A^{k,\text{opt}}}{P}.$$

Finally, substituting $T_{2'}^k = z_k T_2^k$ and $T_{2''}^k = (1-z_k)T_2^k$ into the left-hand side:

$$\mu_k \left(z_k + \frac{1-z_k}{r_k} \right) T_2^k + \frac{1-\mu_k}{r_k} T_3^k \le \frac{A^{k,\text{opt}}}{P},$$

which concludes the proof. □

Lemma 5. *For any $k \in [1, m]$, we have:*

$$\frac{T_1^k}{r_k} + (\mu_k z_k + 1 - z_k)\, T_2^k \le L_f^{k,\text{opt}} \ . \tag{11}$$

Proof. From property (G5) in Table 3, we have:

$$T_2^k = \frac{T_{2'}^k}{z_k} = \frac{T_{2''}^k}{1-z_k} \quad \Rightarrow \quad T_{2'}^k = z_k T_2^k \quad \text{and} \quad T_{2''}^k = (1-z_k)T_2^k.$$

Using property (P1), (P2) and (P3) we have:

$$\frac{T_1^k}{r_k} \le L_{1|f}^{k,\text{opt}}, \qquad \mu_k T_{2'}^k \le L_{2'|f}^{k,\text{opt}}, \qquad T_{2''}^k \le L_{2''|f}^{k,\text{opt}}.$$

Adding the three inequalities, we obtain:

$$\frac{T_1^k}{r_k} + \mu_k T_{2'}^k + T_{2''}^k \le L_{1|f}^{k,\text{opt}} + L_{2'|f}^{k,\text{opt}} + L_{2''|f}^{k,\text{opt}}.$$

Using property (P4):

$$\frac{T_1^k}{r_k} + \mu_k T_{2'}^k + T_{2''}^k \le L_f^{k,\text{opt}}.$$

Substituting $T_{2'}^k = z_k T_2^k$ and $T_{2''}^k = (1-z_k)T_2^k$ gives:

$$\frac{T_1^k}{r_k} + (\mu_k z_k + 1 - z_k)T_2^k \le L_f^{k,\text{opt}},$$

which proves the result. □

Based on the results of Lemmas 4 and 5, we can now derive an upper bound on the makespan of the online scheduling algorithm as shown below.

Theorem 1. *Given $r_m = R$, the maximum ratio among all tasks in the instance, we have:*

$$\frac{T}{T^{\text{opt}}} \le \frac{1}{\mu_m} = \frac{2R+1+\sqrt{4R^2+1}}{2} < 2R+1 \ . \tag{12}$$

Therefore, Fair *is $(2R+1)$-competitive.*

Proof. Recall that $T^k = T_1^k + T_2^k + T_3^k$, and it verifies $T = \sum_k T^k$ (since the corresponding intervals partition the schedule). We multiply both sides by $\frac{1-\mu_k}{r_k}$. Applying Lemma 4 to eliminate $\frac{1-\mu_k}{r_k}T_3^k$ gives:

$$\frac{1-\mu_k}{r_k}T^k \le \frac{1-\mu_k}{r_k}T_1^k + \left(\frac{1-\mu_k - z_k\mu_k r_k - (1-z_k)\mu_k}{r_k}\right)T_2^k + \frac{A^{k,\text{opt}}}{P} \ .$$

We now divide both sides by $(1-\mu_k)$ and apply Lemma 5 (Equation (11)) to remove the $\frac{T_1^k}{r_k}$ term. This gives:

$$\frac{T^k}{r_k} \le \left(\frac{1-2\mu_k + z_k\mu_k - z_k\mu_k r_k}{(1-\mu_k)r_k} - \mu_k z_k + z_k - 1\right)T_2^k + L_f^{k,\text{opt}} + \frac{1}{1-\mu_k}\cdot\frac{A^{k,\text{opt}}}{P} \ .$$

Define the function:

$$f(z_k) = \frac{1-2\mu_k + z_k\mu_k - z_k\mu_k r_k}{(1-\mu_k)r_k} - \mu_k z_k + z_k - 1.$$

We show that $f(z_k) \le 0$ for all $z_k \in [0,1]$ to upper bound the T_2^k term by 0. The derivative is:

$$f'(z_k) = \frac{\mu_k - \mu_k r_k}{(1-\mu_k)r_k} + (1-\mu_k) = \frac{\mu_k + -\mu_k r_k + (1-\mu_k)^2 r_k}{(1-\mu_k)r_k} = \frac{\mu_k + [(1-\mu_k)^2 - \mu_k]r_k}{(1-\mu_k)r_k}.$$

We use property (G4) from Table 3, namely $\mu_k \le \frac{3-\sqrt{5}}{2}$, and note that for all $\mu_k \le \frac{3-\sqrt{5}}{2}$, we have $(1-\mu_k)^2 - \mu_k \ge 0$, so the numerator is positive. Indeed, the smallest root of the polynomial $(1-x)^2 - x$ is precisely $\frac{3-\sqrt{5}}{2}$, so the expression is non-negative for all μ_k below this threshold. Thus $f'(z_k) \ge 0$ and $f(z_k)$ is increasing on $[0,1]$.

Therefore, $f(z_k) \le f(1)$. Now compute:

$$f(1) = \frac{1-\mu_k - \mu_k r_k}{(1-\mu_k)r_k} - \mu_k = 0 \iff \quad \mu_k r_k(1-\mu_k) = 1 - \mu_k - \mu_k r_k.$$

This yields the quadratic:

$$-\mu_k^2 r_k + (2r_k+1)\mu_k - 1 = 0,$$

whose smallest root is:

$$\frac{2r_k + 1 - \sqrt{(2r_k+1)^2 - 4r_k}}{2r_k} = \mu(r_k) = \mu_k,$$

matching the definition in Equation (7). The equality $f(1) = 0$ is shown, and since f increases, for all $z_k \in [0,1]$, $f(z_k) \le 0$ and we can upperbound the T_2^k term by 0 in the main inequality to simplify simplify the inequality to:

$$T^k \le r_k L_f^{k,\text{opt}} + \frac{r_k}{1-\mu_k}\cdot\frac{A^{k,\text{opt}}}{P}.$$

From (G3), the function $\mu(\cdot)$ is decreasing and from (G2), $\frac{r_k}{1-\mu_k}$ increases with r_k. Using $r_k \le R$, we get:

$$T^k \le R L_f^{k,\text{opt}} + \frac{R}{1-\mu_m}\cdot\frac{A^{k,\text{opt}}}{P}.$$

Summing over k:

$$\begin{aligned} T = \sum_k T^k &\le \sum_k \left(R L_f^{k,\text{opt}} + \frac{R}{1-\mu_m}\cdot\frac{A^{k,\text{opt}}}{P}\right) \\ &= R\sum_k L_f^{k,\text{opt}} + \frac{R}{1-\mu_m}\sum_k \frac{A^{k,\text{opt}}}{P}. \end{aligned}$$

Using properties (P5) and (A5) from Table 3:

$$\sum_k L_f^{k,\mathrm{opt}} \le T^{\mathrm{opt}}, \qquad \sum_k A^{k,\mathrm{opt}} \le T^{\mathrm{opt}} \cdot P,$$

we conclude using (G1), and $\mu_m = \mu(R)$:

$$T \le \Big(R + \frac{R}{1-\mu_m}\Big)T^{\mathrm{opt}} = \frac{T^{\mathrm{opt}}}{\mu_m} = \frac{2R}{2R+1-\sqrt{4R^2+1}}\, T^{\mathrm{opt}} = \frac{2R\big(2R+1+\sqrt{4R^2+1}\big)}{(2R+1)^2 - \big(\sqrt{4R^2+1}\big)^2}\, T^{\mathrm{opt}}$$

$$= \frac{2R\big(2R+1+\sqrt{4R^2+1}\big)}{4R}\, T^{\mathrm{opt}} = \frac{2R+1+\sqrt{4R^2+1}}{2}\, T^{\mathrm{opt}}.$$

Finally, note that $\frac{2R+1+\sqrt{4R^2+1}}{2R} < 2R+1$ is immediate, because $4R^2+1 < (2R+1)^2$ □

Remark 1. *The bound is tight for an instance consisting of a single fully parallelisable task. Indeed,* Fair *would allocate $\lceil \mu(1)P \rceil$ processor, which results in a makespan $\mu(1)$ times larger than the optimal makespan, that allocates all P processors to the task.*

We can now apply the previous result to get a general bound for arbitrary speedup profiles, by showing that we always have $R \le \sqrt{P}+1$

Theorem 2. Fair *is* $(2\sqrt{P}+3)$*-competitive*

Proof. Using Theorem 1, this result is true if for any monotonic task, $R_j \le \sqrt{P}+1$. To show this, we distinguish two cases based on whether $p_j^{\max} \le \sqrt{P}$ or not.

Case 1: $p_j^{\max} \le \lceil \sqrt{P} \rceil$

We choose the allocation $p = p_j^{\max}$. By definition, $t_j(p) = t_j^{\min}$, so:

$$\frac{t_j(p)}{t_j^{\min}} = 1.$$

Moreover, since area is non-decreasing, we have:

$$\frac{a_j(p)}{a_j^{\min}} = \frac{a_j(p_j^{\max})}{a_j(1)} \le p_j^{\max} \le \lceil \sqrt{P} \rceil.$$

Hence:

$$R_j(p) = \max\left(1, \frac{a_j(p)}{a_j^{\min}}\right) \le \lceil \sqrt{P} \rceil \le \sqrt{P}+1.$$

Case 2: $p_j^{\max} > \lceil \sqrt{P} \rceil$

We now choose $p = \lceil \sqrt{P} \rceil$, which satisfies $p \le p_j^{\max}$. Since the task satisfies the monotonic property (Definition 2), $t_j(p) \le t_j(1)$, and it cannot exhibit superlinear speedup. In particular, we have the bound:

$$\frac{t_j(p)}{t_j(p_j^{\max})} \le \frac{p_j^{\max}}{p}$$

Applying this with $p = \lceil \sqrt{P} \rceil$, we obtain:

$$\frac{t_j(p)}{t_j^{\min}} = \frac{t_j(p)}{t_j(p_j^{\max})} \le \frac{p_j^{\max}}{\lceil \sqrt{P} \rceil} \le \frac{P}{\lceil \sqrt{P} \rceil} \le \sqrt{P}$$

.

For the area term:

$$\frac{a_j(p)}{a_j^{\min}} = \frac{p \cdot t_j(p)}{t_j(1)} \le \lceil \sqrt{P} \rceil \cdot \frac{t_j(1)}{t_j(1)} = \lceil \sqrt{P} \rceil \le \sqrt{P}+1.$$

Thus:

$$R_j(p) = \max\left(\frac{t_j(p)}{t_j^{\min}}, \frac{a_j(p)}{a_j^{\min}}\right) \le \sqrt{P}+1.$$

In both cases, we have exhibited a processor allocation $p \in [1, p_j^{\max}]$ such that $R_j(p) \leq \sqrt{P} + 1$. Since $R_j = \min_{p \in [1, p_j^{\max}]} R_j(p)$, it follows that:

$$R_j \leq \sqrt{P} + 1.$$

□

Although this competitive ratio may appear large, we will show in the next subsection that it is asymptotically optimal: no algorithm can achieve a competitive ratio smaller than $\Theta(\sqrt{P})$ in the general case. This lower bound significantly improves upon the previous state of the art [21], which only proved that no algorithm could be $o(\log(\log(n)))$-competitive, where n denotes the number of tasks in the instance. Our construction uses only $n = P$ tasks, hence achieves a much stronger lower bound.

5.4 Absolute Lower Bound for the Monotonic Model

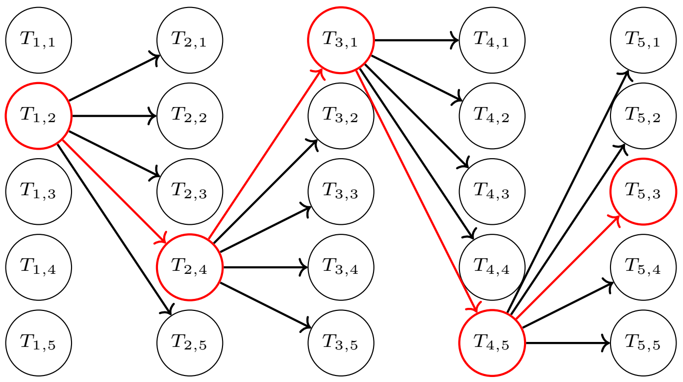


Figure 2: Lower-bound instance for $P = 25$, with 5 layers of 5 tasks. A single critical task per layer (circled in red) has successors in the next layer. Red arrows indicate the critical path formed across layers; all other tasks are independent.

To establish a fundamental lower bound, we construct an adversarial instance consisting of $P = K^2$ processors and $n = P$ tasks, organized into $\sqrt{P}$ layers of $\sqrt{P}$ independent tasks each. Tasks are labeled $\mathcal{T}_{i,j}$, where $i \in \{1, \ldots, \sqrt{P}\}$ denotes the layer index and $j \in \{1, \ldots, \sqrt{P}\}$ the position within the layer.

To introduce dependencies between layers, we designate one critical task per layer. Specifically, for each layer $i \in \{1, \ldots, \sqrt{P} - 1\}$, a single task $\mathcal{T}_{i,j_i}$ is chosen to have successors: it precedes all tasks in the next layer $i+1$, while all other tasks in the layer remain independent. There are no additional precedence constraints.

This construction, illustrated in Figure 2, is designed to model the behavior of an online adversary. Since all tasks in each layer appear identical upon release, the adversary can always pick the task that was completed last in layer i as the critical one $\mathcal{T}_{i,j_i}$, thus enforcing dependencies after the fact and preventing the algorithm from advancing to layer $i+1$ until this specific task has completed.

For example, in Figure 2, we suppose that the algorithm completes task $T_{1,2}$ last in the first layer. The adversary then sets $T_{1,2}$ as the critical task, making all tasks in layer 2 depend on it. The same logic applies in subsequent layers with tasks $T_{2,4}$, $T_{3,1}$, and $T_{4,5}$.

This strategy ensures that the algorithm, regardless of its internal logic, is forced to process the layers sequentially—each layer must be fully processed before discovering the true critical task, which unlocks the next layer. In contrast, an optimal clairvoyant scheduler, aware of all dependencies and critical tasks in advance, would execute only the critical path (highlighted in red in the figure) using all processors, then process the remaining independent tasks in parallel. This large gap between the online and offline capabilities is what gives rise to the lower bound.

We assume a reasonable affine execution time model:

$$t(p) = A - B(p - 1),$$

meaning each additional processor reduces the execution time by a constant B. We set the parameters so that $t(1) = \sqrt{P}$ and $t(P) = 1$, yielding:

$$A = \sqrt{P}, \quad B = \frac{\sqrt{P} - 1}{P - 1}.$$

Theorem 3. *For this instance, any online algorithm has a competitive ratio of at least* $\frac{\sqrt{P}-1}{2}$.

Proof. Since all tasks appear identical when released, an online algorithm cannot distinguish the critical task in each layer. The adversary can thus ensure that the task $\mathcal{T}_{i,j_i}$ finishes last within layer i, delaying the availability of layer $i+1$.

We now analyze the minimum time needed to complete each layer under any algorithm. Two cases may occur:

- **Case 1:** At least one task is allocated at most $\sqrt{P}$ processors. Then its execution time is at least $t(\sqrt{P})$, which we compute explicitly:

$$t(\sqrt{P}) = A - B(\sqrt{P}-1) = \sqrt{P} - \frac{\sqrt{P}-1}{P-1}\cdot(\sqrt{P}-1) \geq \sqrt{P}-1.$$

- **Case 2:** All tasks receive more than $\sqrt{P}$ processors. In this case, execution time is limited not by individual task durations, but by the total available work (area). Since area is increasing with processor count (by monotonicity), the total area of a layer is at least:

$$\sqrt{P}\cdot a(\sqrt{P}) = \sqrt{P}\cdot\sqrt{P}\cdot t(\sqrt{P}) = P\cdot t(\sqrt{P}).$$

  Hence, even with full processor utilization, the layer takes at least:

$$\frac{P\cdot t(\sqrt{P})}{P} = t(\sqrt{P}) \geq \sqrt{P}-1.$$

In both cases, the completion time of each layer is at least $\sqrt{P}-1$, and since there are $\sqrt{P}$ layers, the total execution time satisfies:

$$T \geq \sqrt{P}\cdot(\sqrt{P}-1) = P - \sqrt{P}.$$

Now consider a feasible offline schedule. It first executes the $\sqrt{P}$ critical tasks $\mathcal{T}_{i,j_i}$ sequentially using all P processors (each in time $t(P)=1$), for a total of $\sqrt{P}$ time units. Then, the remaining $P-\sqrt{P}$ tasks can be executed in parallel, each on a single processor, since there are at least $P-\sqrt{P}$ available processors. These tasks have execution time $t(1)=\sqrt{P}$, so the second phase takes at most $\sqrt{P}$ time.

Hence, the total duration of the offline schedule is:

$$T^{\text{opt}} \leq \sqrt{P}+\sqrt{P} = 2\sqrt{P}.$$

Combining this with the earlier lower bound $T \geq P-\sqrt{P}$, we obtain:

$$\frac{T}{T^{\text{opt}}} \geq \frac{P-\sqrt{P}}{2\sqrt{P}} = \frac{\sqrt{P}-1}{2}.$$

□

With these general upper and lower bounds now in place, the remaining task is to determine how large the parameter R can be under the classical speedup models of interest. The next section instantiates this parameter for those models and, by plugging the resulting values into Theorem 1, derives closed-form competitive ratios for FAIR.

6 Special Cases for Different Speedup Models

This section tailors the general framework of Section 5 to the concrete speedup models most frequently studied in the literature. In the first subsection we instantiate the parameter R for each model and, by plugging the resulting values into Theorem 1, obtain explicit competitive ratios for FAIR. The second subsection then argues that these bounds are essentially tight: any substantial improvement seems out of reach, suggesting that our algorithm and analysis is near-optimal.

6.1 Deriving Competitive Ratios

To derive a competitive ratio for a given speedup model, the idea is conceptually straightforward. For each model M, we determine the smallest constant R_M such that, for any task following the model, there exists a processor allocation p satisfying

$$\frac{t_j(p)}{t_j^{\min}} \leq R_M \quad \text{and} \quad \frac{a_j(p)}{a_j^{\min}} \leq R_M.$$

Substituting R_M into Theorem 1 immediately yields a competitive bound of $\frac{2R_M+1+\sqrt{4R_M^2+1}}{2}$ for Fair. Although our objective function differs from that of [21], the overall approach is similar and allows several previously derived results to be reused.

Lemma 6. *For any task that follows the Roofline speedup model, there exists a processor allocation that achieves a ratio no greater than $R_{\text{Roo}} = 1$. Therefore,* Fair *is $\frac{3+\sqrt{5}}{2} < 2.62$-competitive when all tasks follow the Roofline model.*

Proof. This result is standard and follows easily. If a task j follows the Roofline model, its execution time is given by $t(p) = \frac{w}{\min(p,\bar{p})}$, where $w > 0$ and $\bar{p} \in \{1, \dots, P\}$. Setting the processor allocation to $p = \bar{p}$ yields the minimum execution time $t_j^{\min} = \frac{w}{\bar{p}}$. The corresponding area is $a_j(p) = p \cdot t(p) = w$ for all $p \leq \bar{p}$, so the minimum area is achieved with $a_j^{\min} = w$. Hence, $R_j(p) = 1$, and we conclude that $R_{\text{Roo}} = 1$. Plugging this into Equation (12), $T \leq \frac{2R+1+\sqrt{4R^2+1}}{2} T^{\text{opt}}$. yields the claimed competitive ratio. □

Remark 2. *The derived competitive ratio is tight in this setting; the proof is omitted for brevity.*

Lemma 7. *For any task that follows the communication model, there exists a processor allocation that achieves a ratio no greater than $R_{\text{Com}} = \sqrt{2}$. Therefore,* Fair *is $(2+\sqrt{2}) < 3.42$-competitive when all tasks follow the communication model.*

Proof. This proof is inspired by the analysis in [21]. If a task j follows the communication model, its execution time is given by

$$t(p) = \frac{w}{p} + c(p-1) = c\left(\frac{w'}{p} + p - 1\right),$$

with $w' > 0$ and $c > 0$. This function is decreasing on $[1, \sqrt{w'}]$ and increasing on $[\sqrt{w'}, \infty)$.

We first assume $P > w'$. Let $p^{\max}$ denote the number of processors that minimizes the task's execution time, i.e., $t(p^{\max}) = t^{\min}$. Clearly, $\lfloor\sqrt{w'}\rfloor \leq p^{\max} \leq \lceil\sqrt{w'}\rceil$. Also, the minimum area is obtained with one processor: $a^{\min} = a(1) = cw'$.

Define $f(w', p) = \frac{t(p)}{t^{\min}} = \frac{\frac{w'}{p}+p-1}{\frac{w'}{p^{\max}}+p^{\max}-1}$. We show that $f(w', p)$ is non-decreasing with respect to w' in the interval $w' \geq p^2$, by computing the partial derivative:

$$\begin{aligned}\frac{\partial f(w',p)}{\partial w'} &= \frac{\frac{1}{p}\left(\frac{w'}{p^{\max}} + p^{\max} - 1\right) - \frac{1}{p^{\max}}\left(\frac{w'}{p} + p - 1\right)}{\left(\frac{w'}{p^{\max}} + p^{\max} - 1\right)^2} \\ &= \frac{\frac{p^{\max}-1}{p} - \frac{p-1}{p^{\max}}}{\left(\frac{w'}{p^{\max}} + p^{\max} - 1\right)^2}.\end{aligned}$$

This is non-negative when $p \leq p^{\max}$, which is true for $w' \geq p^2$. The continuity of f is straightforward, hence it is non-decreasing. We now consider four cases:

Case 1: $w' \leq 2 + 2\sqrt{2}$, set $p = 1$.

$$\frac{a(p)}{a^{\min}} = \frac{a(1)}{a(1)} = 1,$$

$$\frac{t(p)}{t^{\min}} = f(w', 1) \leq f(2+2\sqrt{2}, 1) = \frac{2+2\sqrt{2}}{\frac{2+2\sqrt{2}}{2}+1} = \frac{2+2\sqrt{2}}{2+\sqrt{2}} = \frac{(2+2\sqrt{2})(2-\sqrt{2})}{2} = \sqrt{2},$$

Indeed, for $w' = 2+2\sqrt{2}$, $p^{\max} = 2$.
Case 2: $2+2\sqrt{2} < w' \le 16$, set $p = 2$.

$$\frac{a(p)}{a^{\min}} = \frac{c(w'+2)}{cw'} = 1 + \frac{2}{w'} < 1 + \frac{1}{1+\sqrt{2}} = 1 + \frac{\sqrt{2}-1}{(1+\sqrt{2})(\sqrt{2}-1)} = \sqrt{2},$$

$$\frac{t(p)}{t^{\min}} = f(w',2) = \frac{\frac{w'}{2}+1}{\frac{w'}{p^{\max}} + p^{\max} - 1} \le f(16,2) = \frac{\frac{16}{2}+1}{\frac{16}{4}+4-1} = \frac{9}{7} < \sqrt{2},$$

Indeed, for $w' = 16$, $p^{\max} = 4$.
Case 3: $16 < w' \le 49$, set $p = 3$.

$$\frac{a(p)}{a^{\min}} = \frac{c(w'+p(p-1))}{cw'} = 1 + \frac{6}{w'} \le 1 + \frac{6}{16} = \frac{22}{16} < \sqrt{2},$$

$$\frac{t(p)}{t^{\min}} = f(w',3) = \frac{\frac{w'}{3}+2}{\frac{w'}{p^{\max}} + p^{\max} - 1} \le f(49,3) = \frac{\frac{49}{3}+2}{\frac{49}{7}+7-1} = \frac{55}{39} < \sqrt{2},$$

Indeed, for $w' = 49$, $p^{\max} = 7$.
Case 4: $w' > 49$. We use the lower bound $t^{\min} \ge c(2\sqrt{w'}-1)$ (attained by a non-integer allocation), and set: $p = \left\lfloor \sqrt{\frac{w'}{3}} + \frac{1}{2} \right\rfloor$. This yields:

$$\frac{a(p)}{a^{\min}} = \frac{c(w'+p(p-1))}{cw'} \le 1 + \frac{1}{w'}\left(\sqrt{\frac{w'}{3}}+\frac{1}{2}\right)\left(\sqrt{\frac{w'}{3}}-\frac{1}{2}\right) \le 1 + \frac{1}{w'}\frac{w'}{3} = \frac{4}{3},$$

$$\frac{t(p)}{t^{\min}} \le \frac{c\left(\frac{w'}{\sqrt{\frac{w'}{3}}-\frac{1}{2}} + \sqrt{\frac{w'}{3}}\right)}{c(2\sqrt{w'}-1)} = \frac{\left(\frac{\sqrt{w'}}{\frac{1}{\sqrt{3}}-\frac{1}{2\sqrt{w'}}} + \sqrt{\frac{w'}{3}}\right)}{(2\sqrt{w'}-1)} = \frac{1}{2-\frac{1}{\sqrt{w'}}}\left(\frac{1}{\frac{1}{\sqrt{3}}-\frac{1}{2\sqrt{w'}}} + \frac{1}{\sqrt{3}}\right).$$

This function is decreasing in w', and using $w' > 49$, we get:

$$\frac{t(p)}{t^{\min}} \le \frac{1}{2-\frac{1}{7}}\left(\frac{1}{\frac{1}{\sqrt{3}}-\frac{1}{14}} + \frac{1}{\sqrt{3}}\right) \approx 1.38 < \sqrt{2}.$$

If $P \le \sqrt{w'}$. In this case, either the previously chosen value of p exceeds P, in which case we set $p = P$ instead. As shown in the earlier analysis, this allocation minimizes the execution time, and the area decreases compared to our earlier derivation—so both bounds still hold. Otherwise, we retain the same value of p as before, which keeps both time and area unchanged. However, the minimum execution time is now larger than in the case $P > \sqrt{w'}$, so our bound on $\frac{t(p)}{t_{\min}}$ remains valid.

In all cases, we conclude that $R_{\text{COM}} = \sqrt{2}$, and substituting this into Equation 12 gives:

$$\frac{T}{T^{\text{opt}}} = \frac{2R_{\text{COM}} + 1 + \sqrt{4R_{\text{COM}}^2 + 1}}{2} = \frac{2\sqrt{2}+1+\sqrt{9}}{2} = 2 + \sqrt{2}.$$

□

Remark 3. *The derived competitive ratio is tight in this setting; the proof is omitted for brevity.*

Lemma 8. *For any task that follows the Amdahl speedup model, there exists a processor allocation that achieves Power a ratio no greater $R_{\text{AMD}} = 2$. Therefore,* Fair *is $\frac{5+\sqrt{17}}{2} < 4.562$-competitive when all tasks follow the Amdahl model.*

Proof. This result is well known and straightforward. If a task follows the Amdahl model, its execution time can be written $t(p) = \frac{w}{p} + d$, with $w > 0$ and $d > 0$. The minimum execution time satisfies $t^{\min} > d$, and the minimum area is obtained using a single processor, i.e., $a^{\min} = a(1) = w + d$.

We set $p = \min\left(\left\lceil \frac{w}{d} \right\rceil, P\right)$. Since $p \leq \left\lceil \frac{w}{d} \right\rceil$, we have

$$\frac{a(p)}{a^{\min}} = \frac{w + dp}{w + d} \leq \frac{w + d\left(\frac{w}{d} + 1\right)}{w + d} = \frac{w + d + w}{w + d} \leq 2.$$

If $p = \left\lceil \frac{w}{d} \right\rceil$ (and thus $p \leq P$), then

$$\frac{t(p)}{t^{\min}} \leq \frac{\frac{w}{\frac{w}{d}} + d}{d} \leq 2.$$

Otherwise $p = P$, so $t(p) = t^{\min}$ and consequently $\frac{t(p)}{t^{\min}} = 1 < 2$.

Hence, this gives $R_{\text{AMD}} = 2$, and substituting this value into Equation 12 yields the desired competitive ratio. □

Remark 4. *The derived competitive ratio is tight in this setting; the proof is omitted for brevity.*

Lemma 9. *For any task that follows the General speedup model, there exists a processor allocation that achieves Power a ratio no greater $R^{\text{GEN}} < 2.018$. Therefore,* Fair *is 4.6-competitive when all tasks follow the General model.*

Proof. If a task follows the General model, its execution time can be written $t(p) = \frac{w}{\min(p,\bar{p})} + d + c(p-1) = c\left(\frac{w'}{\min(p,\bar{p})} + d' + p - 1\right)$, with $w' > 0$, $d' > 0$, $c > 0$, and $\bar{p} \geq 1$. We distinguish two cases:

Case 1: $w' \leq 800$. Appendix B shows that there exists a $p \in [1, 8]$ such that $t(p)/t_{\min} \leq 2.003$ and $a(p)/a_{\min} \leq 2$. The proof proceeds by an exhaustive subcase analysis; for example, if $w' \in [185, 485]$ and $d' \in [80, 170]$, the choice $p = 3$ is valid. The detailed enumeration (13 subcases) is omitted here.

Case 2: $w' > 800$. Using a result from [21], choosing $p = \min\left(\left\lfloor \frac{w'+d'}{\sqrt{w'+d'}} + \frac{1}{2} \right\rfloor, \bar{p}\right)$ guarantees $t(p)/t_{\min} \leq 2$ and $a(p)/a_{\min} \leq \frac{1}{1-\frac{1}{2\sqrt{w'}}} + 1$. The latter expression decreases with w', and for $w' \geq 800$ we obtain $a(p)/a_{\min} < 2.018$.

In all cases, the ratio is at most $R^{\text{GEN}} < 2.018$. Plugging this into Equation 12 shows that the competitive ratio is below 4.6. □

Remark 5. *As with Amdahl, we believe that a sharper analysis would give $R_{\text{GEN}} = 2$, in which case the derived competitive ratio would be identical to that of Amdahl and thus tight in this setting.*

Lemma 10. *For any task that follows the Power Communication model, there exists a processor allocation that achieves Power a ratio no greater $R^{\text{PowCom}} < 2.00424$. Therefore,* Fair *is 4.57-competitive when all tasks follow the Power Communication model.*

Proof. If a task follows the Power Communication model, its execution time can be written $t(p) = \frac{w}{p} + cp^{\gamma} = c\left(\frac{w'}{p} + p^{\gamma}\right)$, with $w' > 0$ and $0 < \gamma \leq 1$. Allowing p to take non-integer values, a quick analysis shows that the function $t(p)$ is minimized at $p^{\star} = \left(\frac{w'}{\gamma}\right)^{\frac{1}{\gamma+1}}$; with $t_{\min} \geq c\,(\gamma+1)\left(\frac{w'}{\gamma}\right)^{\frac{\gamma}{\gamma+1}}$

We first assume $P > p^{*}$. We distinguish two cases:

Case 1: $w' \leq 8$. Appendix C shows that there exists a $p \in [1, 8]$ such that $t(p)/t_{\min} \leq 2$ and $a(p)/a_{\min} \leq 2$. The proof proceeds by an exhaustive subcase analysis; for example, if $w' \in [3, 4]$ and $\gamma \leq 0.1$, choosing $p = 4$ is valid. The detailed enumeration (12 subcases) is omitted here.

Case 2: $w' \geq 8$ and $0 \leq \gamma \leq 0.1$ We set $p = \left\lceil w'^{\frac{1}{\gamma+1}} \right\rceil \leq p^{*} < P$. Then

$$\frac{t(p)}{t^{\min}} = \frac{\frac{w'}{p} + p^{\gamma}}{t^{\min}} \leq \frac{\frac{w'}{w'^{\frac{1}{\gamma+1}}} + \left(w'^{\frac{1}{\gamma+1}} + 1\right)^{\gamma}}{(\gamma+1)\left(\frac{w'}{\gamma}\right)^{\frac{\gamma}{\gamma+1}}} \leq \frac{w'^{\frac{\gamma}{\gamma+1}} + w'^{\frac{\gamma}{\gamma+1}}\left(1 + \frac{1}{w'^{\frac{1}{\gamma+1}}}\right)^{\gamma}}{w'^{\frac{\gamma}{\gamma+1}}\left(\frac{\gamma+1}{\gamma^{\frac{\gamma}{\gamma+1}}}\right)}$$

$$\leq \frac{\gamma^{\frac{\gamma}{\gamma+1}}}{\gamma+1}\left(1 + \left(1 + \frac{1}{w'^{\frac{1}{\gamma+1}}}\right)^{\gamma}\right) \leq \frac{1^{\frac{\gamma}{\gamma+1}}}{\gamma+1}\left(1 + (1+1)^{\gamma}\right) = \frac{2^{\gamma}+1}{\gamma+1},$$

and, because $f(\gamma) = 2^{\gamma} + 1$ is convex while $g(\gamma) = 2(\gamma+1)$ is affine, with $f(0) \leq g(0)$ and $f(0.1) \leq g(0.1)$, we obtain $\frac{t(p)}{t^{\min}} \leq 2$.

$$\frac{a(p)}{a_{\min}} = \frac{w' + p^{\gamma+1}}{w'+1} \leq \frac{w' + \left(w'^{\frac{1}{\gamma+1}} + 1\right)^{\gamma+1}}{w'+1} = \frac{w' + w'\left(1 + w'^{-\frac{1}{\gamma+1}}\right)^{\gamma+1}}{w'+1} \leq \frac{w' + w'\left(1 + w'^{-\frac{1}{1.1}}\right)^{1.1}}{w'+1}.$$

Recall that for any real exponent a and $|x| < 1$ we have the generalized binomial expansion

$$(1+x)^a = \sum_{k=0}^{\infty} \binom{a}{k} x^k, \qquad \binom{a}{k} = \frac{a(a-1)\cdots(a-k+1)}{k!}.$$

When $1 < a < 2$ the first three coefficients are positive $(\binom{a}{0}, \binom{a}{1}, \binom{a}{2} > 0)$, whereas for every $k \geq 3$ the coefficients alternate in sign and their absolute values decrease monotonically. Hence the tail $\sum_{k=3}^{\infty} \binom{a}{k} x^k$ is non-positive for $0 < x < 1$ (it starts with a negative term and the remaining terms form an alternating series of diminishing magnitude). Therefore truncating after the quadratic term provides an upper bound:

$$\left(1 + w'^{-\frac{1}{1.1}}\right)^{1.1} \leq 1 + 1.1\, w'^{-\frac{1}{1.1}} + 0.055\, w'^{-\frac{2}{1.1}} \leq 1 + 1.1\, w'^{-\frac{1}{1.1}} + 0.1\, w'^{-1}.$$

Hence

$$\frac{a(p)}{a_{\min}} \leq \frac{w' + w'\left(1 + 1.1\, w'^{-\frac{1}{1.1}} + 0.1\, w'^{-1}\right)}{w'+1} \leq \frac{2w' + 1.1\, w'^{\frac{1}{11}} + 0.1}{w'+1} = 2 + \frac{1.1\, w'^{\frac{1}{11}} - 1.9}{w'+1}.$$

If $w' \leq \left(\frac{1.9}{1.1}\right)^{11}$ the fraction is non-positive, so $\frac{a(p)}{a_{\min}} \leq 2$. Otherwise,

$$\frac{a(p)}{a_{\min}} \leq 2 + 1.1\, w'^{-\frac{10}{11}} \leq 2 + \left(\tfrac{1.9}{1.1}\right)^{-10} < 2.00424.$$

Thus, for all $w' \geq 8$ and $0 \leq \gamma \leq 0.1$,

$$\frac{t(p)}{t^{\min}} \leq 2 \quad \text{and} \quad \frac{a(p)}{a_{\min}} < 2.00424,$$

Case 3: $w' \geq 8$ and $\gamma > 0.1$. We set $p = \left\lfloor w'^{\frac{1}{\gamma+1}} \right\rfloor \leq p^* < P$.

$$\frac{t(p)}{t^{\min}} = \frac{\frac{w'}{p} + p^{\gamma}}{t^{\min}} \leq \frac{\frac{w'}{w'^{\frac{1}{\gamma+1}} - 1} + \left(w'^{\frac{1}{\gamma+1}}\right)^{\gamma}}{(\gamma+1)\left(\frac{w'}{\gamma}\right)^{\frac{\gamma}{\gamma+1}}} = \frac{\gamma^{\frac{\gamma}{\gamma+1}}}{\gamma+1} \cdot \frac{\frac{w'^{\frac{\gamma}{\gamma+1}}}{1 - w'^{-\frac{1}{\gamma+1}}} + w'^{\frac{\gamma}{\gamma+1}}}{w'^{\frac{\gamma}{\gamma+1}}} \leq \frac{1}{\gamma+1}\left(\frac{1}{1 - w'^{-\frac{1}{\gamma+1}}} + 1\right).$$

The right-hand side decreases with w', so we may replace w' with 8. The first factor that involves γ decreases with γ, whereas the second increases. Define

$$f(a,b) = \frac{1}{a+1}\left(\frac{1}{1 - 8^{-\frac{1}{b+1}}} + 1\right).$$

If $a \leq \gamma \leq b$, then $\frac{t(p)}{t^{\min}} \leq f(a,b)$, because we upper-bound the decreasing factor by a and the increasing one by b. A numerical evaluation gives

$$f(0.1, 0.15) < 2, \qquad f(0.15, 0.4) < 2, \qquad f(0.4, 1) < 2,$$

and therefore $\frac{t(p)}{t^{\min}} < 2$.

Finally,

$$\frac{a(p)}{a_{\min}} = \frac{w' + p^{\gamma+1}}{w'+1} \leq \frac{w' + \left(w'^{\frac{1}{\gamma+1}}\right)^{\gamma+1}}{w'+1} \leq \frac{2w'}{w'+1} < 2.$$

If $P \leq p^*$. In this case, either the previously chosen value of p exceeds P, in which case we set $p = P$ instead. As shown in the earlier analysis, this allocation minimizes the execution time, and the area decreases compared to our earlier derivation—so both bounds still hold. Otherwise, we retain the same value of p as before, which keeps both time and area unchanged. However, the minimum execution time is now larger than in the case $P > p^*$, so our bound on $\frac{t(p)}{t_{\min}}$ remains valid.

In all cases, $R^{\text{PowCom}} < 2.00424$ yields a competitive ratio strictly below 4.57 when substituting in Equation 12.

□

Remark 6. *As with Amdahl, we believe that a sharper analysis would give* $R_{\text{PowCom}} = 2$*, in which case the derived competitive ratio would match that of Amdahl and be tight in this setting.*

We have derived competitive ratios for all considered speedup models, which are very close to those previously obtained by algorithms tailored to each model, even though our algorithm is general. In the next subsection, we explain why these results are unlikely to be improved upon, unless through fundamentally new techniques.

6.2 Potential for Improvements

To argue why further progress is unlikely without fundamentally new ideas, we first frame a broad family of algorithms.

The LALIST paradigm. We say that an algorithm belongs to *Local-Allocation List Scheduling* (LALIST) when it satisfies two conditions:

(i) **Eager list scheduling.** Whenever at least one task is ready and sufficient processors are idle, the algorithm schedules some ready task immediately. The particular priority rule is irrelevant, but deliberate idling is forbidden.

(ii) **Local processor allocation.** The processor allocation chosen for a task depends only on that task's speedup profile $t(p)$. Consequently, two identical tasks receive identical allocations, independent of the current system state or of other tasks in the queue.

In the online setting, our knowledge is inherently limited to the task that has just been released; no information about successors that have yet to be revealed is available. Consequently, previous work such as [5, 21] determine the allocation of each task solely from that task's speedup function, exactly as required by condition (ii). Once allocations are fixed, a greedy (ASAP) list scheduler seems to be the only reasonable option, because any deliberate idling can be exploited by an adversary to extend the makespan. This is why condition (i) was introduced.

To further explain the validity of these assumptions, let's study the best algorithms when the entire DAG is known. They all follow *two-phase approach*:

- (1) analyze the full graph to assign a processor allocation to every task;
- (2) run a greedy list schedule on those fixed allocations [24, 7, 20, 16].

While allocations in Phase (1) can rely on global DAG properties in the offline setting, such information is unavailable online. Crucially, even though offline algorithms assign allocations differently depending on global DAG structure—accelerating critical-path tasks and possibly slowing tasks without successors—they never adapt allocations based on instantaneous system load during the scheduling phase. Since online algorithms lack knowledge of the DAG structure, condition (ii) naturally emerges: tasks with identical speedup profiles must receive identical allocations, because there is no reliable structural information to justify differentiating them. The only way left to treat tasks differently would be to look at the system state at each moment. But even the best offline algorithms, despite knowing the whole DAG in advance, avoid this kind of state-based tweaking, because it would make the policy much more complicated and too hard to analyze. This motivates the strong assumption in the LALIST paradigm, where task allocations depend only on their individual time functions.

Intuitively, allowing the allocation to react to the instantaneous load may appear advantageous (e.g., "use all processors if only one task is ready"). Yet these locally sound decisions can backfire under adversarial conditions, for instance, when a large number of tasks arrive just after allocating too many processors to a single task. Even in the offline setting, this possibility is hard to anticipate. For these reasons, only genuinely new breakthrough ideas (offline or online) could lead to an algorithm that outperforms all LALIST heuristics, and we think such a breakthrough is very unlikely.

Known limits of LALIST. In [21] the authors established tight lower bounds on the competitive ratio of any LALIST algorithm. For the Roofline, communication, and Amdahl models, those bounds are met by model-specific algorithms; as shown in Table 4 [4] . No LALIST strategy—regardless of its priority rule—can do better.

[4] For the General model, the algorithm presented in [21] actually achieves a better bound than Fair. Using the refined analysis, it can be shown to be 4.58-competitive.

Table 4: Summary of parameters and competitive ratios for different speedup models. The best possible bounds are rounded down; bounds from our heuristics are rounded up. Tight bounds are shown in red.

Model M	[5]	[21]	Fair	Lower Bound on Best LALIST Heuristic
Roofline (Roo)	2.62	2.62	2.62	$\frac{2}{3-\sqrt{5}} > 2.61$
Comm. (Com)	3.61	3.40	3.41	$\frac{18}{23-\sqrt{313}} > 3.39$
Amdahl (Amd)	4.74	4.55	4.56	$\frac{2}{1-\sqrt{8\sqrt{2}-11}} > 4.54$
General (Gen)	5.72	4.63(4.58)*	4.6	$\frac{2}{1-\sqrt{8\sqrt{2}-11}} > 4.54$
Extended COM (COM2)			4.57	$\frac{2}{1-\sqrt{8\sqrt{2}-11}} > 4.54$

Where Fair stands. Strictly speaking, Fair is not in LALIST, because each allocation also depends on the global ratio r observed so far. However, a more clairvoyant variant, given the model-dependent constant R^M in advance, could directly initialize r to R^M instead of 1. Such algorithm would be identical to Fair except for Line 2 of Algorithm 2, where the initial ratio is set. The analysis would then proceed unchanged with $k = 1$ since no further updates of r (and hence of μ) would occur. This algorithm would belong to LALIST, achieving exactly the bounds of Fair shown in Table 4.

Thus the deviations of Fair from LALIST are not due to online state adaptation, but merely to learning each task's speedup model on the fly. That Fair still achieves a competitive ratio within 1% of the best LALIST algorithms — including those that know the speedup model perfectly from the start, such as those in [5, 21] — shows how little room for improvement remains.

In summary, Fair almost matches the best algorithm that follows the LALIST paradigm. Even if a clairvoyant initialization of the ratio were possible, it would not yield a better bound: the non-clairvoyant version of Fair already achieves a ratio very close to the best attainable by any LALIST algorithm, including those that know all task profiles in advance (such as the algorithms in [5, 21]). To improve the ratios further, one would need techniques that violate at least one of the defining LALIST conditions. Yet, even with full knowledge of the DAG, no better strategy than the classic two-phase "allocate everything then list-schedule" approach is known. Hence, discovering an online algorithm that outperforms all LALIST strategies still seems far out of reach.

After deriving and discussing all these theoretical guarantees, we now turn to an empirical validation. Section 7 explains how we generate workloads, which heuristics we test, and how we measure performance.

7 Experiments

In this section, we first describe the workloads (§D.1), then the heuristics (§D.2), and finally the experimental protocol (§D.3), before discussing the outcome of the experiments (§7.4).

7.1 Instance Generation

Task generation. Our study evaluates the scheduling heuristics on the five speedup models defined in Section 3.1: Roofline, Communication, Amdahl, Power Communication, and General. For every instance each task is generated independently as follows:

- **Total work** w_j — drawn in seconds from the task-generation framework; it equals $t_j(1)$ for every model.
- **Parallelism bound** $p_j^{\max}$ — uniform in $[1, 512]$.
- **Sequential fraction ratio** ρ_d — log-uniform in $[10^{-3}, 10^{-0.5}]$ (i.e. $\rho_d = 10^{\mathrm{Unif}(-3,-0.5)}$); the absolute sequential part used in Amdahl-type terms is $d_j = \rho_d\, w_j$.
- **Communication ratio** ρ_c — log-uniform in $[10^{-6}, 10^{-2}]$ (i.e. $\rho_c = 10^{\mathrm{Unif}(-6,-2)}$); the communication coefficient in the time functions is $c_j = \rho_c\, w_j$.
- **Power exponent** γ_j — uniform in $[0, 1]$ (only used in the power-communication model).

Sampling ρ_d and ρ_c draws the sequential part and the communication cost relative to each task's total work: a large task (w_j high) is therefore more likely to carry a proportionally larger sequential or communication component.

Graph generation. Task-dependency graphs are produced with `daggen`,[5] which creates synthetic DAGs organised in layers. Edges connect a task in one layer to tasks in subsequent layers; if the *Jump* parameter exceeds 1, an edge may skip intermediate layers.

The following topology parameters control the structure of each graph:

- **Density** $\in [0, 1]$ (default 0.5) — controls how many edges are created between consecutive layers. A value near 0 yields a very sparse DAG (few dependencies, almost chain-like), whereas a value near 1 produces a much denser graph with many inter-layer edges.
- **Fat** $\in [0, 1]$ (default 0.5) — width of the DAG. Low values create thin, chain-like graphs; high values yield wide, fork–join-like structures.
- **Regular** $\in [0, 1]$ (default 0.5) — uniformity of the task distribution across layers.
- **Jump** $\in [1, 9]$ (default 5) — maximum number of layers a dependency may skip, controlling the length of execution paths.
- **Number of tasks** $n \in [500, 5000]$ (default 2500).
- **Number of processors** $P \in [124, 1024]$ (default 512).

7.2 Heuristics

For every speedup model we compare five allocation policies:

- **ICPP22** — heuristics presented in [5] (labelled *ICPP22* in all tables). Again, the heuristic is specific to the model tested.
- **TOPC24** — heuristics presented in [21] (labelled *TOPC24*), also specific to the model tested. [6]
- **Fair** — our single, model-agnostic heuristic.
- **minTime** — allocate $p_j = p_j^{\max}$, minimising each task's execution time, and schedule tasks ASAP.
- **minArea** — allocate $p_j = 1$, minimising each task's processor-time area, and schedule tasks ASAP.

These policies differ only in their processor-allocation rule. They are orthogonal to the priority rule used to select which ready task is launched next; we therefore evaluate four queue-ordering strategies:

- **FIFO** (default) — first in, first out.
- **procs** — task requiring the largest allocation first [7].
- **area** — task with the largest area $a(p) = t(p) \times p$ first.
- **length** — task with the largest execution time $t(p)$ first.

7.3 Experimental Protocol

Param	**Default**	**Test Values**	**Type**	**Description**
n	2500	$[500, \dots, 5000]$	Scale	Number of tasks
P	512	$[124, \dots, 1024]$	System	Available processors
Priority	FIFO	[FIFO, procs . . .]	Policy	Queue-ordering strategy
Density	0.5	$[0, \dots, 1]$	Graph	Edge probability
Fat	0.5	$[0, \dots, 1]$	Graph	Width/depth ratio
Regular	0.5	$[0, \dots, 1]$	Graph	Degree uniformity
Jump	5	$[1, \dots, 9]$	Graph	Maximum jump distance between layers

Table 5: Experimental parameters and their default values.

[5] Commit `f3e8b6c` of `https://github.com/frs69wq/daggen`.

[6] For the Power Communication model we extend the allocation rules of both [5] and [21] using the analysis of Section 6; the resulting variants follow the spirit of the original papers.

[7] For Fair, we very slightly modify the Algorithm 2 by precomputing the p', line 11, before the loop to ensure the queue is correctly ordered

All default settings and test ranges are summarised in Table 7. For every speedup model and every parameter summarized in the table, we perform one experiment that isolates the impact of that parameter on heuristic quality. To do so, we normalize the makespan by a simple lower bound on the best makespan achievable.

Practical lower bound. The analysis in Section 5 compares each heuristic against the optimal makespan T^{opt}, whose two classical components are $\frac{A^{\text{opt}}}{P}$ (area bound) and C^{opt} (critical-path bound). Unfortunately, computing either A^{opt} or the allocations that realise C^{opt} is NP-complete.

For large-scale empirical comparisons we therefore use a looser, polynomial Lower Bound:

$$L \;=\; \max\Bigl(\tfrac{A^{\min}}{P},\; C^{\min}\Bigr),$$

where

$$A^{\min} \;=\; \sum_{j=1}^{n} a_j(1), \qquad C^{\min} \;=\; \max_{f} \sum_{j\in f} t_j\bigl(p_j^{\max}\bigr).$$

- $A^{\min}$ is the total area if every task runs on a single processor, which is the minimum possible area for the instance.
- $C^{\min}$ is the length of the longest path when each task is executed with its allocation $p_j^{\max}$ that minimizes $t_j(\,\cdot\,)$.

Both quantities are readily computable in $O(|E|+|V|)$ time and clearly satisfy $A^{\min} \le A^{\text{opt}}$ and $C^{\min} \le C^{\text{opt}}$, hence $L \le T^{\text{opt}}$. While L is less tight than the theoretical bound [8], it is sufficient for comparing heuristics.

The experimental protocol is as follows:

1. Set every parameter to its default value except the one under study, which is swept over all of its **Test Values**.
2. For each setting, generate 50 random instances and simulate them with all heuristics. The performance metric is the ratio $\frac{T}{L}$, where L is the lower bound on the optimal makespan previously defined.
3. Convert the 50 ratio values per heuristic into a box-plot. The box spans the 25th–75th percentiles; the horizontal black line marks the median; a star denotes the average. Whiskers extend to the 10th and 90th percentiles, and outliers beyond that range are shown as small circles. However, all the detailed results of the experiments for each pair, using these box-plots, is not reported in the main part of the paper.
4. Produce two summary tables: (i) the average ratio for each (heuristic, speedup model) pair plus the overall mean; (ii) the worst ratio observed for each pair. Although, in principle, an empirical ratio could exceed the proven competitive guarantee (which compares to the true optimum), this never occurred; even after normalising by L, all ratios remained below the theoretical limit. Given the looseness of our bound, it is likely that no instance was even close to the worst case scenario.

Complete numerical results and figures are provided in Appendix D.

Table 6: Average Values for Each Model and Heuristic

Model	ICPP22	TOPC24	Fair	minTime	minArea
Roofline	1.28	1.28	1.28	1.52	11.28
Communication	1.75	1.38	1.38	2.97	10.22
Amdahl	2.32	1.98	2.01	21.31	7.88
General	2.74	1.50	1.60	4.86	5.37
PowerCom	1.52	1.46	1.35	2.84	12.76
Average	1.92	1.52	1.52	6.70	9.50

[8]Lower bounds tighter than the one we adopt and computable in polynomial time also exist [20], but evaluating them would largely dominate the runtime of the experiment.

7.4 Summary of Results

Table 6 compares Fair with earlier model-specific heuristics and two simple baselines (minTime and minArea) over all speed-up models. Across the board, Fair matches the best specialised algorithm in each model while vastly outperforming the naive baselines.

- *Roofline model*: Fair attains an average competitive ratio of 1.28, identical to the best prior heuristic and far below the baselines. It is as expected, since all heuristics are the same for this specific case since they can minimize both time and area.
- *Communication model*: Fair reaches 1.38, essentially tying the specialised algorithm presented published in TOPC24 presented in ICPP22.
- *Amdahl model*: Fair yields 2.01, on par with the 1.98 ratio of the tailored TOPC24 algorithm.
- *General model*: Fair achieves 1.60, only slightly higher than the best specialised ratio (1.50).
- *Power Communication model*: Fair obtains 1.35 and in fact outperforms the prior state of the art (1.46).

Averaged over all models, Fair delivers a competitive ratio of 1.52, tying the best specialised heuristic and standing an order of magnitude ahead of the minTime and minArea baselines. These empirical findings confirm the theoretical guarantees: a single universal scheduler can achieve the same high performance previously attainable only with model-specific tuning. Note that we only experimented when all tasks belong to the same model, because the competitors may not be extended for mixed workflows.

8 Conclusion

This work set out to answer whether a single online scheduler could rival model-specific algorithms for moldable tasks with arbitrary speedup functions. The answer is affirmative.

We introduced the Fair algorithm, proved a general competitive-ratio bound for arbitrary monotonic speed-up profiles, and specialised the analysis to the Roofline, Communication, Amdahl, General and Power Communication models. For the first three models, Fair attains the optimal competitive ratio permitted by existing lower bounds; for the remaining models it comes within a few per cent of the best possible heuristic that respect two intuitive conditions. Hence, further improvement in terms of competitive ratio seems unlikely. On the practical side, extensive simulation with tens of thousands of synthetic DAGs showed that Fair consistently matches or surpasses the best tailored heuristics in practice, while the naive minTime and minArea policies perform an order of magnitude worse.

These results demonstrate that universal online scheduling for moldable tasks is both achievable and efficient. Since the theory already seems very optimized, the most promising next step is integrating Fair into a production HPC or cloud scheduler, exercising it on real job traces, and measuring its impact on throughput and utilization. Additional directions include extending the universal approach to malleable tasks or to objectives beyond makespan, such as energy efficiency or user fairness.

In short, Fair provides a near-optimal, model-agnostic solution to online moldable scheduling, closing a gap between specialized theory and practical applicability.

References

[1] Kunal Agrawal, Charles E. Leiserson, and Jim Sukha. Executing task graphs using work-stealing. In *IPDPS*, pages 1–12, 2010.

[2] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS'67*, pages 483–485, 1967.

[3] Anne Benoit, Valentin Le Fèvre, Lucas Perotin, Padma Raghavan, Yves Robert, and Hongyang Sun. Resilient scheduling of moldable jobs on failure-prone platforms. In *IEEE Cluster*, 2020.

[4] Anne Benoit, Valentin Le Fèvre, Lucas Perotin, Padma Raghavan, Yves Robert, and Hongyang Sun. Resilient scheduling of moldable parallel jobs to cope with silent errors. *IEEE Transactions on Computers*, 71(07):1696–1710, 2022.

[5] Anne Benoit, Lucas Perotin, Yves Robert, and Hongyang Sun. Online scheduling of moldable task graphs under common speedup models. In *Proceedings of the 51st International Conference on Parallel Processing*, ICPP '22, New York, NY, USA, 2023. Association for Computing Machinery.

[6] Louis-Claude Canon, Loris Marchal, Bertrand Simon, and Frédéric Vivien. Online scheduling of task graphs on heterogeneous platforms. *IEEE Trans. Parallel Distributed Syst.*, 31(3):721–732, 2020.

[7] Chi-Yeh Chen and Chih-Ping Chu. A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints. *IEEE Trans. Parallel Distrib. Syst.*, 24(8):1479–1488, 2013.

[8] Richard A. Dutton and Weizhen Mao. Online scheduling of malleable parallel jobs. In *PDCS*, pages 136–141, 2007.

[9] Dror G. Feitelson and Larry Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer, 1996.

[10] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. Optimal on-line scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4):393–411, 1998.

[11] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[12] Jessen T. Havill and Weizhen Mao. Competitive online scheduling of perfectly malleable jobs with setup times. *European Journal of Operational Research*, 187:1126–1142, 2008.

[13] K. Jansen and F. Land. Scheduling monotone moldable jobs in linear time. In *IPDPS*, pages 172–181, 2018.

[14] Klaus Jansen. A $(3/2 + \epsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *SPAA*, pages 224–235, 2012.

[15] Klaus Jansen and Hu Zhang. Scheduling malleable tasks with precedence constraints. In *SPAA*, page 86–95, 2005.

[16] Klaus Jansen and Hu Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algorithms*, 2(3):416–434, 2006.

[17] Berit Johannes. Scheduling parallel jobs to minimize the makespan. *J. of Scheduling*, 9(5):433–452, 2006.

[18] Theodore Johnson, Timothy A. Davis, and Steven M. Hadfield. A concurrent dynamic task graph. *Parallel Computing*, 22(2):327–333, 1996.

[19] Nathaniel Kell and Jessen Havill. Improved upper bounds for online malleable job scheduling. *J. of Scheduling*, 18(4):393–410, 2015.

[20] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. In *ESA*, pages 146–157, 2001.

[21] Lucas Perotin and Hongyang Sun. Improved online scheduling of moldable task graphs under common speedup models. *ACM Trans. Parallel Comput.*, 11(1), March 2024.

[22] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[23] John Turek, Joel L. Wolf, and Philip S. Yu. Approximate algorithms scheduling parallelizable tasks. In *SPAA*, 1992.

[24] Qingzhou Wang and Kam Hoi Cheng. A heuristic of scheduling parallel tasks and its analysis. *SIAM J. Comput.*, 21(2):281–294, 1992.

[25] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009.

[26] Deshi Ye, Danny Z. Chen, and Guochuan Zhang. Online scheduling of moldable parallel tasks. *J. of Scheduling*, 21(6):647–654, 2018.

A Appendix : Proof that the models are monotonic

Lemma 11. *All tasks belonging to speedup models introduced in this paper are monotonic on* $[1, p_j^{\max}]$

Proof. Since the Roofline, Communication and Amdahl models are all special cases of the General Model, we only need to show the results for the General Model and the Power Communication Model. We must show that the given models satisfy the two conditions of the monotonic property:

1. General Model. Recall that for the General Model, the execution time is given by:

$$t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)} + d_j + c_j(p_j - 1).$$

Non-increasing execution time:
For $p_j \leq \bar{p}_j$, we have:

$$t_j(p_j) = \frac{w_j}{p_j} + d_j + c_j(p_j - 1).$$

The derivative with respect to p_j is:

$$t'_j(p_j) = -\frac{w_j}{p_j^2} + c_j.$$

This derivative cancels when:

$$p_j = \sqrt{\frac{w_j}{c_j}}\ .$$

Let us define:

$$p_j^* = \arg \min_{p \in \left\{ \left\lfloor \sqrt{\frac{w_j}{c_j}} \right\rfloor, \left\lceil \sqrt{\frac{w_j}{c_j}} \right\rceil \right\}} t_j(p)$$

as the integer value that minimizes $t_j(p)$ among the two integers closest to the real-valued optimum.

We distinguish two cases:

- If $\min(P, \bar{p}_j) \leq p_j^*$, then the execution time strictly decreases on $[1, \min(P, \bar{p}_j)]$, and the minimum is attained at $p_j^{\max} = \min(P, \bar{p}_j)$.
- Otherwise, $t_j(p)$ decreases until p_j^* and increases afterward. Thus, the execution time is non-increasing on $[1, p_j^{\max}]$, where $p_j^{\max} = \min\left(p_j^*, \bar{p}_j, P\right)$.

Non-decreasing area:
The area is defined by $a_j(p_j) = p_j \cdot t_j(p_j)$.
For $p_j \leq \bar{p}_j$:

$$a_j(p_j) = p_j \left(\frac{w_j}{p_j} + d_j + c_j(p_j - 1) \right) = w_j + p_j d_j + c_j p_j (p_j - 1).$$

The derivative is:

$$a'_j(p_j) = d_j + c_j(2p_j - 1).$$

With $c_j > 0$ and $p_j \geq 1$, this derivative is always non-negative, so $a_j(p_j)$ is non-decreasing on $[1, \bar{p}_j]$.

For $p_j > \bar{p}_j$, recall that $t_j(p_j)$ increases linearly, so:

$$a_j(p_j) = p_j \cdot \left(\frac{w_j}{\bar{p}_j} + d_j + c_j(p_j - 1) \right),$$

which is also strictly increasing as it is a convex quadratic function. Thus, $a_j(p_j)$ is non-decreasing on the whole domain.

Hence, the General Model satisfies both monotonic conditions.

2. Power Communication model Recall the execution-time expression

$$t_j(p_j) \;=\; \frac{w_j}{p_j} \;+\; c_j\, p_j^{\gamma_j}, \qquad w_j > 0,\ c_j > 0,\ 0 < \gamma_j \leq 1.$$

Shape of the execution-time curve. Our goal is to show that $t_j(p_j)$ is either monotonically increasing, or it first decreases and then increases, as the processor allocation p_j grows.

Taking the derivative gives

$$t_j'(p_j) \;=\; -\frac{w_j}{p_j^2} \;+\; c_j\gamma_j\, p_j^{\gamma_j-1}.$$

Multiplying by p_j^2 (strictly positive) lets us track the sign more easily:

$$g(p_j) \;=\; p_j^2 t_j'(p_j) \;=\; -\,w_j \;+\; c_j\gamma_j\, p_j^{\gamma_j+1}.$$

Hence, $\operatorname{sign}\bigl(t_j'(p_j)\bigr) = \operatorname{sign}\bigl(g(p_j)\bigr)$.

Key properties of g.

- At $p_j = 1$: $g(1) = c_j\gamma_j - w_j$.
- As $p_j \to \infty$: $g(p_j) \to \infty$ because $\gamma_j + 1 > 0$.
- Derivative: $g'(p_j) = c_j\gamma_j(\gamma_j+1)\, p_j^{\gamma_j} > 0$; hence g is strictly increasing.

Two regimes follow.

1. **Always-increasing time ($c_j\gamma_j \geq w_j$).**
   When $g(1) \geq 0$, monotonicity of g implies $g(p_j) \geq 0$ for every $p_j \geq 1$. Consequently $t_j'(p_j) \geq 0$ and $t_j(p_j)$ grows with the processor allocation.

2. **Decrease-then-increase ($c_j\gamma_j < w_j$).**
   Here $g(1) < 0$ but $g(p_j)$ eventually becomes positive. Because g is continuous and strictly increasing, there exists a unique
   $$p_j^* > 1 \quad \text{such that} \quad g(p_j^*) = 0.$$

Thus, $t_j(p_j)$ either increases or decreases then increases over $p_j \geq p_j^{\max} = \arg\min_{p\in\{\lfloor p_j^*\rfloor, \lceil p_j^*\rceil\}} t_j(p)$.

Monotonicity of the area. The processor-time area is

$$a_j(p_j) \;=\; p_j\, t_j(p_j) \;=\; w_j \;+\; c_j\, p_j^{\gamma_j+1}.$$

Differentiating,

$$a_j'(p_j) \;=\; c_j(1+\gamma_j)\, p_j^{\gamma_j} \;\geq\; 0,$$

because $c_j > 0$ and $\gamma_j \geq 0$. Hence the area is non-decreasing in the processor allocation, as required.

□

B Appendix: Complete Proof of General Ratio

Lemma 12. *For any task that follows the General speedup model, there exists a processor allocation that achieves a ratio no greater than $R^{\text{GEN}} < 2.018$. Consequently,* FAIR *is 4.6-competitive when all tasks follow the General model.*

Proof. If a task follows the General model, its execution time can be written

$$t(p) = \frac{w}{\min(p,\bar{p})} + d + c(p-1) = c\left(\frac{w'}{\min(p,\bar{p})} + d' + p - 1\right),$$

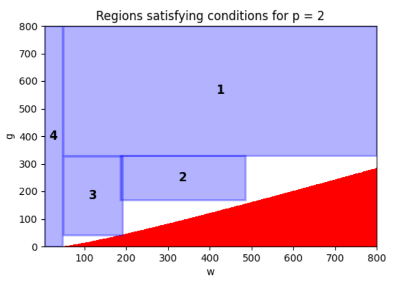
with $w' > 0$, $d' > 0$, $c > 0$, and $\bar{p} \geq 1$. Allowing p to take non-integer values, and assuming $\bar{p}$ is unbounded, the function $t(p)$ is minimized at $p^\star = \sqrt{w'}$; hence

$$t^{\min} \;\geq\; c\bigl(2\sqrt{w'} + d' - 1\bigr).$$

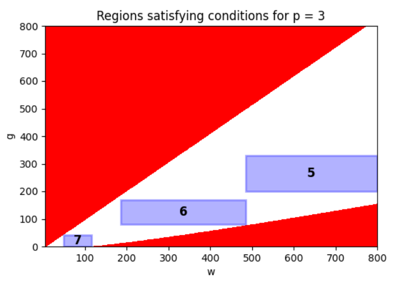
(The bound also holds when $\bar{p} < p^\star$, since then the minimum occurs at $p = \bar{p}$ and is therefore larger.) The minimum area is obtained with one processor: $a^{\min} = a(1) = c(w' + d')$.

Let $p^{\max}$ denote the (integer) processor count that minimizes the execution time, i.e. $t(p^{\max}) = t^{\min}$. Either $p^{\max} = \bar{p}$ or $\lfloor\sqrt{w'}\rfloor \leq p^{\max} \leq \lceil\sqrt{w'}\rceil$.

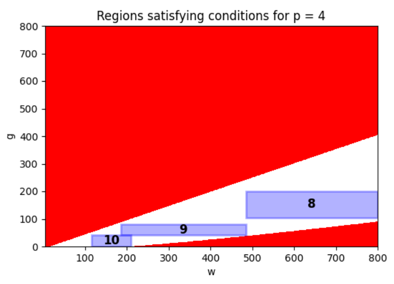
We analyze four cases.



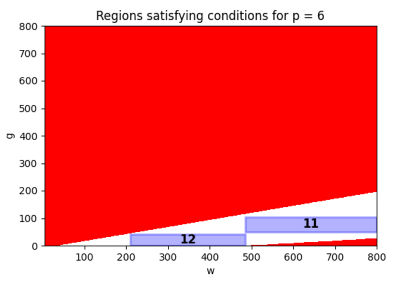
(a) $p = 2$



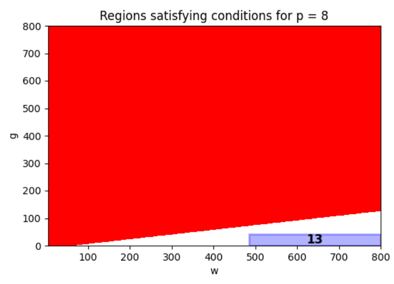
(b) $p = 3$



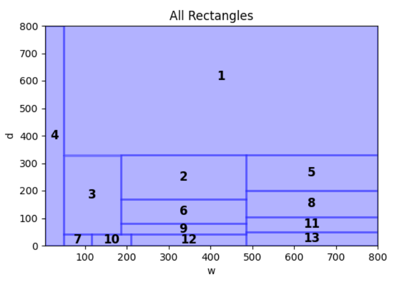
(c) $p = 4$



(d) $p = 6$



(e) $p = 8$



(f) All Rectangles

Figure 3: Comparative plots for different values of p.

Case 1: $w' \leq 4$ or $\bar{p} = 1$. Then $p^{\max} \leq 2$; setting $p = 1$ yields

$$\frac{a(p)}{a^{\min}} = 1, \qquad \frac{t(p)}{t^{\min}} \leq 2.$$

Case 2: $4 < w' \leq 800$ and $d' > 800$. Choosing $p = 1$ gives $a(p) = a^{\min}$. Since $t^{\min} > d' > w'$, $t(p) = w + d < 2\,t^{\min}$.

Case 3: We first assume that $\bar{p}$ is unbounded. Define

$$\alpha(w', d', p) = \frac{a(w', d', p)}{a_{\min}(w', d')} = \frac{w' + pd' + p(p-1)}{w' + d'} = 1 + \frac{(p-1)(d'+p)}{w' + d'}.$$

The function $\alpha(w', d', p)$ is decreasing in w'. Differentiating with respect to d' gives

$$\frac{\partial \alpha(w', d', p)}{\partial d'} = \frac{(w'-p)(p-1)}{(w'+d')^2} > 0 \quad \text{whenever } w' > p.$$

Next, set

$$\beta(w', d', p) = \frac{t(w', d', p)}{t_{\min}(w', d')} \geq \frac{\frac{w'}{p} + d' + p - 1}{\frac{w'}{p^{\max}} + d' + p^{\max} - 1},$$

for any $p^{\max} \geq p$. With p fixed and $p < p^{\max}$, the ratio β increases with w' (a larger w' implies more parallel work and thus a larger $\frac{t}{t_{\min}}$) and decreases with d' for the exact same reason as long as $p < p^{\max}$. This can be verified formally and is omitted here for brevity. Finally, note that $p^{\max}$ itself naturally increases with w' (more parallelism) and decreases with d' (less parallelism).

Therefore, suppose we are given a pair (w', d') such that $w_1 \leq w' \leq w_2$, $d_2 \leq d' \leq d_1$, and a value p satisfying $p^{\max}(w_1, d_1) \geq p$, $\alpha(w_1, d_1, p) \leq 2$, and $\beta(w_2, d_2, p) \leq 2$.

Then $\alpha(w', d', p) \leq 2$ and $\beta(w', d', p) \leq 2$ for the entire rectangle $\left[w_1, w_2\right] \times \left[d_2, d_1\right]$.

Figure 5 illustrates this idea. For each $p \in \{2, 3, 4, 6, 8\}$ we color in red the regions where $\alpha(w', d', p) > 2$ (upper-left) and where $\beta(w', d', p) > 2$ (lower-right), using a Python script. We then subdivide the domain ($w' \in [4, 800]$, $d' \in [0, 800]$) into 13 sub-rectangles that don't intersect the red zones, assigning a single value of p to each; for example, sub-rectangle 5 uses $p = 3$. The existence of an allocation of an allocation respecting the conditions can be viewed in the figure, but to confirm that a sub-rectangle avoids the red zones without relying on the Python script, it suffices to check:

Rectangle	(w_1, γ_1)	(w_2, γ_2)	p	$p^{\max}(w_1, \gamma_1, p)$	$f(w_1, \gamma_1, p)$	$g(w_2, \gamma_2, p)$
1	$(49, 800)$	$(800, 330)$	2	7	1.94	1.90
2	$(185, 330)$	$(485, 170)$	2	14	1.64	1.94
3	$(49, 330)$	$(185, 42)$	2	7	1.88	1.99
4	$(4, 800)$	$(49, 0)$	2	2	2.00	1.96
5	$(485, 330)$	$(800, 200)$	3	22	1.82	1.83
6	$(185, 170)$	$(485, 80)$	3	14	1.97	1.98
7	$(49, 42)$	$(115, 0)$	3	7	1.99	1.97
8	$(485, 200)$	$(800, 105)$	4	22	1.89	1.92
9	$(185, 80)$	$(485, 42)$	4	14	1.95	1.95
10	$(115, 42)$	$(210, 0)$	4	11	1.88	1.98
11	$(485, 105)$	$(800, 50)$	6	22	1.94	1.78
12	$(210, 42)$	$(485, 0)$	6	14	1.95	1.99
13	$(485, 50)$	$(800, 0)$	8	22	1.76	1.93

Figure 4: Verification of extreme values in sub-areas

- the upper-left corner for α: $\alpha(w_1, d_1, p) \leq 2$ (in Figure 5, this is the closest point to the upper-left redzone corresponding to which corresponds to excessive area),
- the lower-right corner for β: $\beta(w_2, d_2, p) \leq 2$ (in Figure 5, this is the closest point to the lower-right redzone which corresponds to excessive execution time),
- that $p \leq p^{\max}$ at the upper-left corner: $p^{\max}(w_1, d_1) \geq p$ (as this inequality will hold true in all the rectangle if it is true for the top-left corner).

All of these checks are summarised in the table in Figure 4.

We have thus shown that, when $\bar{p}$ is unbounded, there exists a processor count p' such that $\frac{t(p')}{t_{\min}} \leq 2.018$ and $\frac{a(p')}{a_{min}} \leq 2.018$. If instead $\bar{p} < p'$, we simply set $p = \bar{p} = p^{\max}$; this choice reduces the area and yields $t(p) = t_{\min}$.

Case 4: $w' > 800$. This case is treated in the main body of the paper.

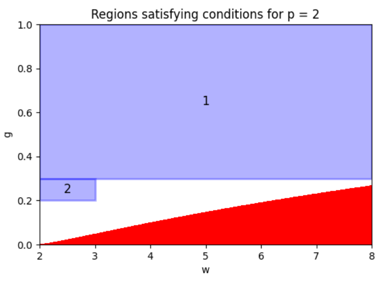
In every case the ratio is bounded by $R^{\text{GEN}} < 2.018$. Substituting this value into Equation (12) gives a competitive ratio strictly below 4.6. $\square$

C Appendix : Complete Proof of Power Communication Ratio

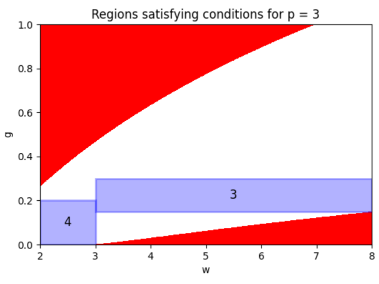
Lemma 13. *For any task that follows the Power Communication model, there exists a processor allocation that achieves Power a ratio no greater $R^{\text{PowCom}} < 2.00424$. Therefore,* Fair *is 4.57-competitive when all tasks follow the Power Communication model.*

Proof. If a task follows the Power Communication model, its execution time can be written $t(p) = \frac{w}{p} + cp^{\gamma} = c\left(\frac{w'}{p} + p^{\gamma}\right)$, with $w' > 0$ and $0 < \gamma \leq 1$. Allowing p to take non-integer values, a quick analysis shows that the function $t(p)$ is minimized at $p^{\star} = \left(\frac{w'}{\gamma}\right)^{\frac{1}{\gamma+1}}$; with $t_{\min} \geq (\gamma+1)\left(\frac{w'}{\gamma}\right)^{\frac{\gamma}{\gamma+1}}$. In this appendix, we assume $P > p^{*}$, since the other case is treated in the main part of the paper. We distinguish three cases:

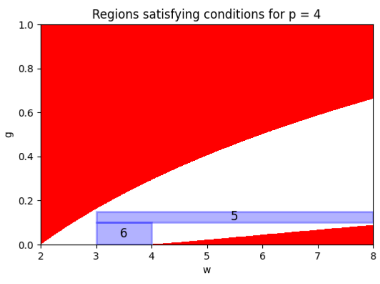
Case 1: $w \leq 2$. If $p^{\max} = 1$, then we choose $p = 1$ and get $\frac{a(p)}{a_{\min}} = \frac{t(p)}{t_{\min}} = 1$. Otherwise, we choose $p = 2$. Since $t(2) \leq t(1)$, we have $a(2) \leq 2a_{\min}$. Finally, $t(2) = c\left(\frac{w'}{2} + 2^{\gamma}\right) \leq c(1+2^{\gamma}) \leq c\left(1 + (p^{\max})^{\gamma}\right) \leq c + t_{\min}$. As $t_{\min} \geq c$, it follows that $\frac{t(p)}{t_{\min}} \leq 2$.



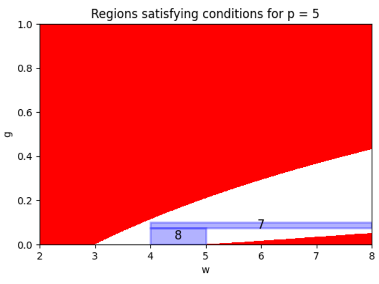
(a) $p = 2$



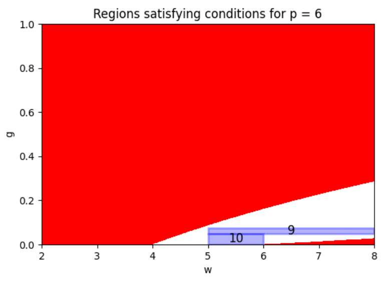
(b) $p = 3$



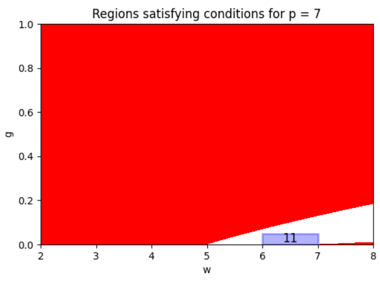
(c) $p = 4$



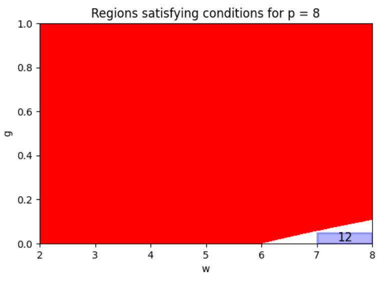
(d) $p = 5$



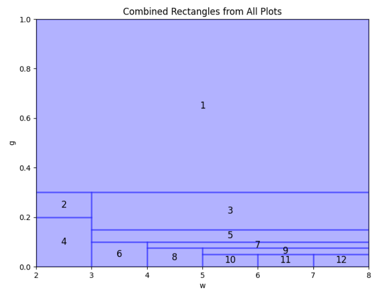
(e) $p = 6$



(f) $p = 7$



(g) $p = 8$



(h) All Rectangles

Figure 5: Comparative plots for different values of p.

Case 2: $2 \leq w' < 8$. Let $\alpha(w',\gamma,p) = \frac{a(w',\gamma,p)}{a_{\min}(\gamma,p)} = \frac{w'+p^{\gamma+1}}{w'+1}$ and $\beta(w',\gamma,p) = \frac{t(w',\gamma,p)}{t_{\min}(\gamma,p)} = \frac{\frac{w'}{p}+p^{\gamma}}{\frac{w'}{p^{\max}}+p^{\max\gamma}}$. The objective is to show that for any pair (w',γ), there exists a p such that $\alpha(w',\gamma,p) \leq 2$ and $\beta(w',\gamma,p) \leq 2$.

We observe that $\alpha(w',\gamma,p) = 1 + \frac{p^{\gamma+1}-1}{w'+1}$ increases with γ and decreases with w'. Similarly, if $p^{\max} \geq p$, with p fixed and $p < p^{\max}$, the ratio $\beta(w',\gamma,p)$ increases with w' (a larger w' implies more parallel work and thus a larger

Rectangle	(w_1, γ_1)	(w_2, γ_2)	p	$p_{\max}(w_1, \gamma_1, p)$	$\alpha(w_1, \gamma_1, p)$	$\beta(w_2, \gamma_2, p)$
1	$(2, 1)$	$(8, 0.3)$	2	1	2.00	1.89
2	$(2, 0.3)$	$(3, 0.2)$	2	4	1.49	1.41
3	$(3, 0.3)$	$(8, 0.15)$	3	6	1.79	1.99
4	$(2, 0.2)$	$(3, 0)$	3	7	1.91	2.00
5	$(3, 0.15)$	$(8, 0.1)$	4	14	1.98	1.92
6	$(3, 0.1)$	$(4, 0)$	4	22	1.90	2.00
7	$(4, 0.1)$	$(8, 0.075)$	5	29	1.97	1.83
8	$(4, 0.075)$	$(5, 0)$	5	40	1.93	2.00
9	$(5, 0.075)$	$(8, 0.05)$	6	50	1.98	1.82
10	$(5, 0.05)$	$(6, 0)$	6	80	1.93	2.00
11	$(6, 0.05)$	$(7, 0)$	7	96	1.96	2.00
12	$(7, 0.05)$	$(8, 0)$	8	111	1.98	2.00

Figure 6: Verification of all extreme values in sub-areas

$\frac{t}{t_{\min}}$) and decreases with γ (since a higher communication cost implies less efficient parallelism). Additionally, $p^{\max}$ increases with w' (more parallelism) and decreases with γ (less parallelism), assuming all other parameters are fixed. This can be verified via a quick analysis, omitted here.

Therefore, we use the same idea as in the previous appendix. If for a pair (w', γ) we are given $w_1 \leq w' \leq w_2$, $\gamma_2 \leq \gamma \leq \gamma_1$, and a p such that $p^{\max}(w_1, \gamma_1) \geq p$, $\alpha(w_1, \gamma_1, p) \leq 2$, and $\beta(w_2, \gamma_2, p) \leq 2$, then we are assured that $\alpha(w', \gamma, p) \leq 2$ and $\beta(w', \gamma, p) \leq 2$.

We further illustrate this with Figure 5: for each $p \in [2, 8]$, we color in red the areas where $\alpha(w', \gamma, p) > 2$ (top-left area) and where $\beta(w', \gamma, p) > 2$ (bottom-right area). We subdivide the region ($w' \in [2, 8]$, $\gamma \in [0, 1]$) into 12 sub-rectangles, each assigned a single value of p. For instance, rectangle 3 is assigned $p = 3$. We can visually see in the figure the existence of a valid choice of p for any pair w', γ, but to confirm that a sub-rectangle avoids the red zones without relying on the script, it suffices to check:

- the upper-left corner for α: $\alpha(w_1, \gamma_1, p) \leq 2$,
- the lower-right corner for β: $\beta(w_2, \gamma_2, p) \leq 2$,
- that $p \leq p^{\max}$ at the upper-left corner: $p^{\max}(w_1, \gamma_1) \geq p$,

All these checks are summarized in the table in Figure 6 (note that the values are rounded, but the entries marked 2.00 are exactly equal to 2).

Case 3: $w' > 8$. This case is treated in the main body of the paper.

In all cases, we conclude that $R^{\text{PowCom}} < 2.00424$, which yields a competitive ratio strictly below 4.57 when substituted into Equation 12. $\square$

D Appendix : Full Experimental Results

For the reader's convenience, we restate the experimental protocol and describe the meaning of the graphs.

D.1 Instance Generation

Task generation. Our study evaluates the scheduling heuristics on the five speedup models defined in Section 3.1: Roofline, Communication, Amdahl, Power Communication, and General. For every instance each task is generated independently as follows:

- **Total work** w_j — drawn in seconds from the task-generation framework; it equals $t_j(1)$ for every model.
- **Parallelism bound** $p_j^{\max}$ — uniform in $[1, 512]$.
- **Sequential fraction ratio** ρ_d — log-uniform in $[10^{-3}, 10^{-0.5}]$ (i.e. $\rho_d = 10^{\text{Unif}(-3,-0.5)}$); the absolute sequential part used in Amdahl-type terms is $d_j = \rho_d \, w_j$.
- **Communication ratio** ρ_c — log-uniform in $[10^{-6}, 10^{-2}]$ (i.e. $\rho_c = 10^{\text{Unif}(-6,-2)}$); the communication coefficient in the time functions is $c_j = \rho_c \, w_j$.
- **Power exponent** γ_j — uniform in $[0, 1]$ (only used in the power-communication model).

Sampling ρ_d and ρ_c draws the sequential part and the communication cost relative to each task's total work: a large task (w_j high) is therefore more likely to carry a proportionally larger sequential or communication component.

Graph generation. Task-dependency graphs are produced with `daggen`,[9] which creates synthetic DAGs organised in layers. Edges connect a task in one layer to tasks in subsequent layers; if the *Jump* parameter exceeds 1, an edge may skip intermediate layers.

The following topology parameters control the structure of each graph:

- **Density** $\in [0, 1]$ (default 0.5) — controls how many edges are created between consecutive layers. A value near 0 yields a very sparse DAG (few dependencies, almost chain-like), whereas a value near 1 produces a much denser graph with many inter-layer edges.
- **Fat** $\in [0, 1]$ (default 0.5) — width of the DAG. Low values create thin, chain-like graphs; high values yield wide, fork–join-like structures.
- **Regular** $\in [0, 1]$ (default 0.5) — uniformity of the task distribution across layers.
- **Jump** $\in [1, 9]$ (default 5) — maximum number of layers a dependency may skip, controlling the length of execution paths.
- **Number of tasks** $n \in [500, 5000]$ (default 2500).
- **Number of processors** $P \in [124, 1024]$ (default 512).

D.2 Heuristics

For every speedup model we compare five allocation policies:

- **ICPP22** — heuristics presented in [5] (labelled *ICPP22* in all tables). Again, the heuristic is specific to the model tested.
- **TOPC24** — heuristics presented in [21] (labelled *TOPC24*), also specific to the model tested. [10]
- **Fair** — our single, model-agnostic heuristic.
- **minTime** — allocate $p_j = p_j^{\max}$, minimising each task's execution time, and schedule tasks ASAP.
- **minArea** — allocate $p_j = 1$, minimising each task's processor-time area, and schedule tasks ASAP.

These policies differ only in their processor-allocation rule. They are orthogonal to the priority rule used to select which ready task is launched next; we therefore evaluate four queue-ordering strategies:

- **FIFO** (default) — first in, first out.
- **procs** — task requiring the largest allocation first [11].
- **area** — task with the largest area $a(p) = t(p) \times p$ first.
- **length** — task with the largest execution time $t(p)$ first.

[9] Commit `f3e8b6c` of `https://github.com/frs69wq/daggen`.

[10] For the Power Communication model we extend the allocation rules of both [5] and [21] using the analysis of Section 6; the resulting variants follow the spirit of the original papers.

[11] For Fair, we very slightly modify the Algorithm 2 by precomputing the p', line 11, before the loop to ensure the queue is correctly ordered

D.3 Experimental Protocol

Param	Default	Test Values	Type	Description
n	2500	$[500, \dots, 5000]$	Scale	Number of tasks
P	512	$[124, \dots, 1024]$	System	Available processors
Priority	FIFO	[FIFO, procs . . .]	Policy	Queue-ordering strategy
Density	0.5	$[0, \dots, 1]$	Graph	Edge probability
Fat	0.5	$[0, \dots, 1]$	Graph	Width/depth ratio
Regular	0.5	$[0, \dots, 1]$	Graph	Degree uniformity
Jump	5	$[1, \dots, 9]$	Graph	Maximum jump distance between layers

Table 7: Experimental parameters and their default values.

All default settings and test ranges are summarised in Table 7. For every speedup model and every parameter summarized in the table, we perform one experiment that isolates the impact of that parameter on heuristic quality. To do so, we normalize the makespan by a simple lower bound on the best makespan achievable.

Practical lower bound. The analysis in Section 5 compares each heuristic against the optimal makespan T^{opt}, whose two classical components are $\frac{A^{\text{opt}}}{P}$ (area bound) and C^{opt} (critical-path bound). Unfortunately, computing either A^{opt} or the allocations that realise C^{opt} is NP-complete.

For large-scale empirical comparisons we therefore use a looser, polynomial Lower Bound:

$$L = \max\!\left(\frac{A^{\min}}{P},\, C^{\min} \right),$$

where

$$A^{\min} = \sum_{j=1}^{n} a_j(1), \qquad C^{\min} = \max_{f} \sum_{j \in f} t_j\!\left(p_j^{\max}\right).$$

- $A^{\min}$ is the total area if every task runs on a single processor, which is the minimum possible area for the instance.
- $C^{\min}$ is the length of the longest path when each task is executed with its allocation $p_j^{\max}$ that minimizes $t_j(\,\cdot\,)$.

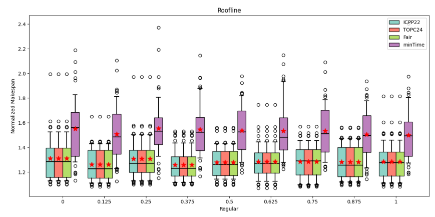
Both quantities are readily computable in $O(|E| + |V|)$ time and clearly satisfy $A^{\min} \le A^{\text{opt}}$ and $C^{\min} \le C^{\text{opt}}$, hence $L \le T^{\text{opt}}$. While L is less tight than the theoretical bound [12], it is sufficient for comparing heuristics.

The experimental protocol is as follows:

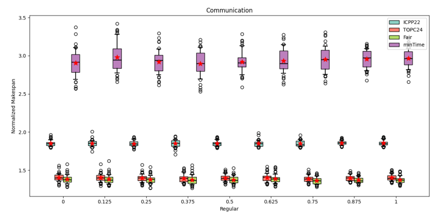
1. Set every parameter to its default value except the one under study, which is swept over all of its **Test Values**.
2. For each setting, generate 50 random instances and simulate them with all heuristics. The performance metric is the ratio $\frac{T}{L}$, where L is the lower bound on the optimal makespan previously defined.
3. Convert the 50 ratio values per heuristic into a box-plot. The box spans the 25th–75th percentiles; the horizontal black line marks the median; a star denotes the average. Whiskers extend to the 10th and 90th percentiles, and outliers beyond that range are shown as small circles. However, all the detailed results of the experiments for each pair, using these box-plots, is not reported in the main part of the paper.
4. Produce two summary tables: (i) the average ratio for each (heuristic, speedup model) pair plus the overall mean; (ii) the worst ratio observed for each pair. Although, in principle, an empirical ratio could exceed the proven competitive guarantee (which compares to the true optimum), this never occurred; even after normalising by L, all ratios remained below the theoretical limit. Given the looseness of our bound, it is likely that no instance was even close to the worst case scenario.

[12]Lower bounds tighter than the one we adopt and computable in polynomial time also exist [20], but evaluating them would largely dominate the runtime of the experiment.

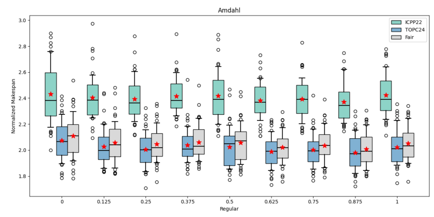
D.4 Detailed Results



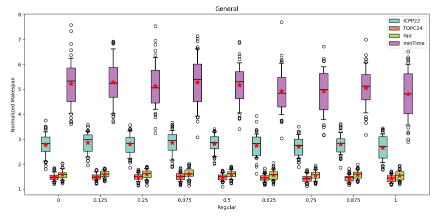
(a) Roofline



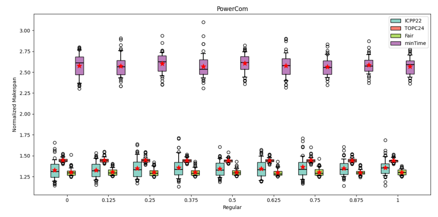
(b) Communication



(c) Amdahl

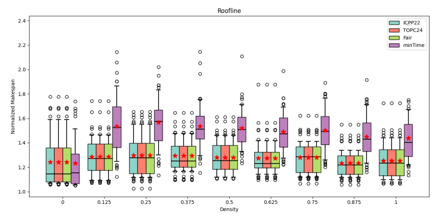


(d) General

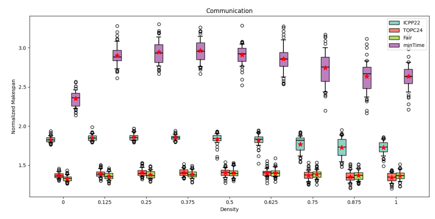


(e) PowerCom

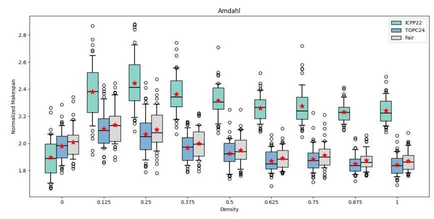
Figure 7: Impact of task distribution regularity (Regular) on scheduling performance.



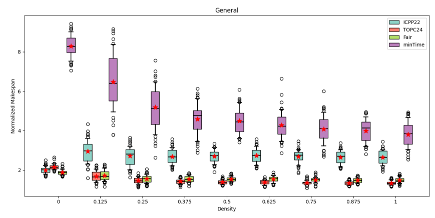
(a) Roofline



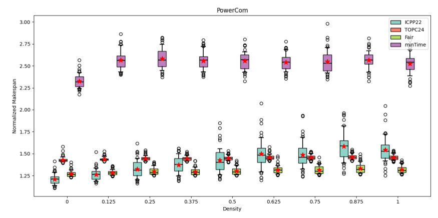
(b) Communication



(c) Amdahl

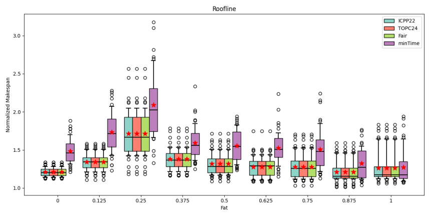


(d) General

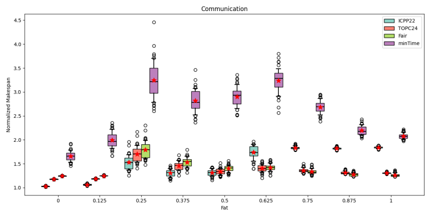


(e) PowerCom

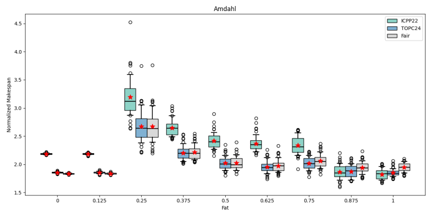
Figure 8: Impact of inter-layer dependency density (Density) on heuristic performance.



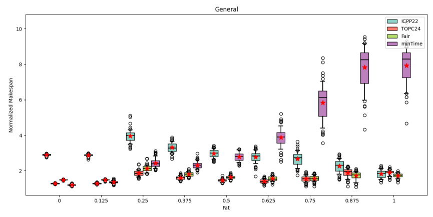
(a) Roofline



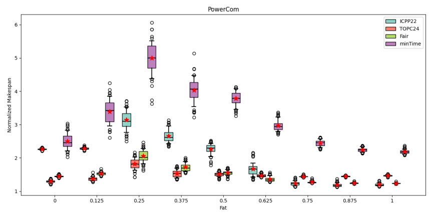
(b) Communication



(c) Amdahl

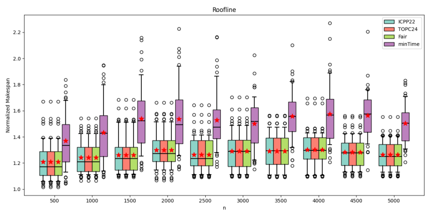


(d) General

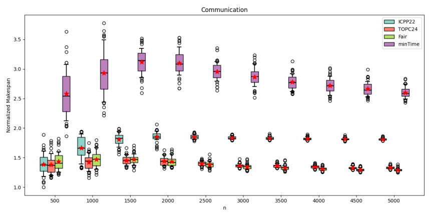


(e) PowerCom

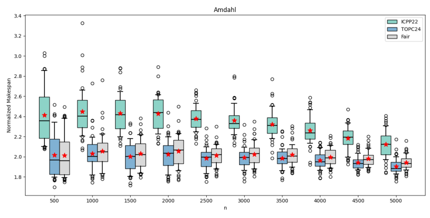
Figure 9: Impact of DAG width-to-depth ratio (Fat) on heuristic efficiency.



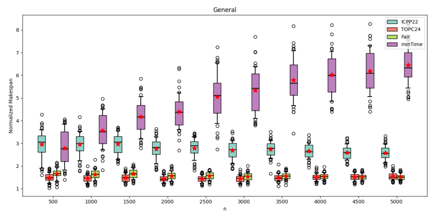
(a) Roofline



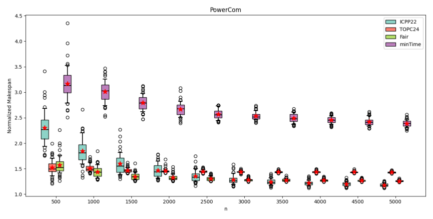
(b) Communication



(c) Amdahl

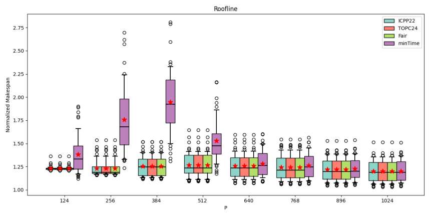


(d) General

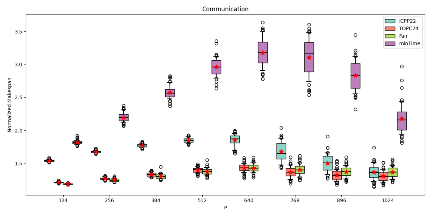


(e) PowerCom

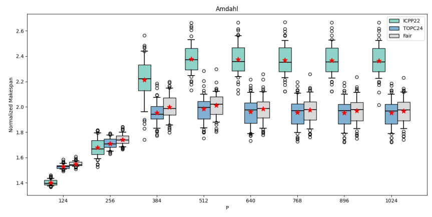
Figure 10: Impact of total number of tasks (n) on scheduling performance.



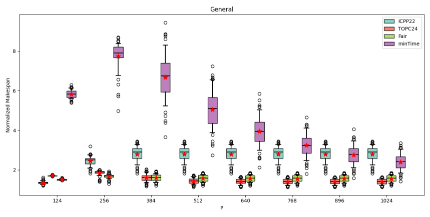
(a) Roofline



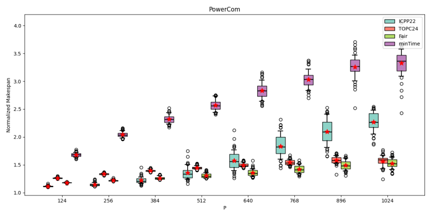
(b) Communication



(c) Amdahl

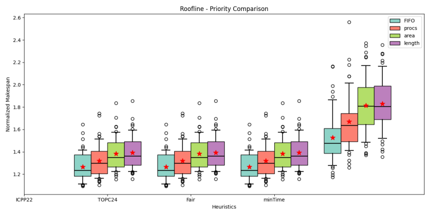


(d) General

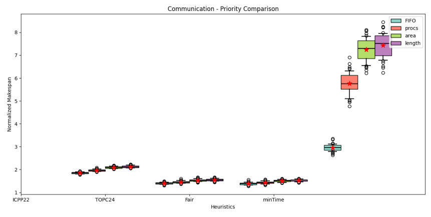


(e) PowerCom

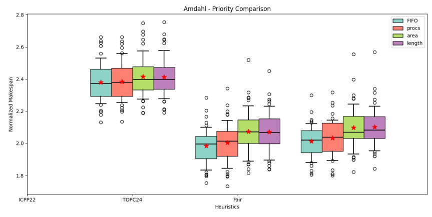
Figure 11: Impact of available processors (P) on makespan ratio.



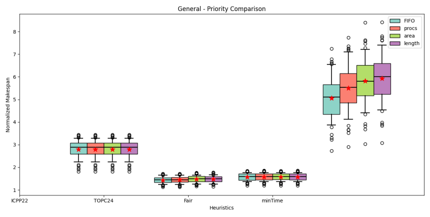
(a) Roofline



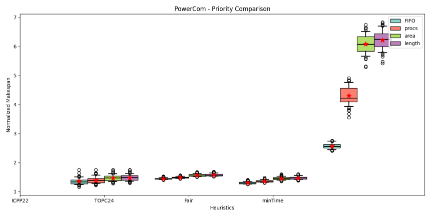
(b) Communication



(c) Amdahl

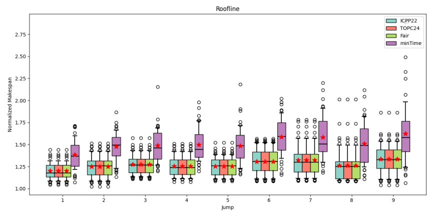


(d) General

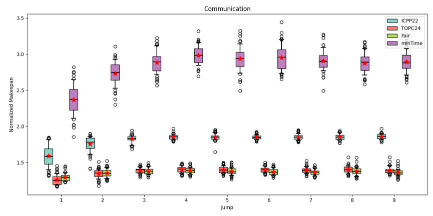


(e) PowerCom

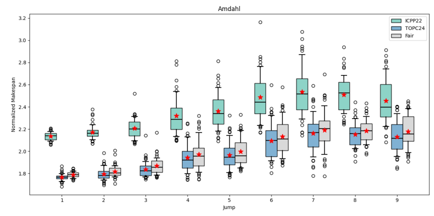
Figure 12: Impact of queue ordering strategy (Priority) on performance.



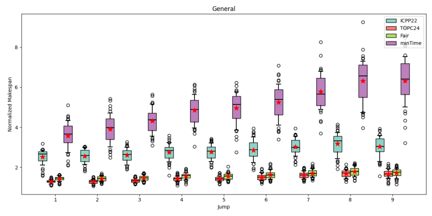
(a) Roofline



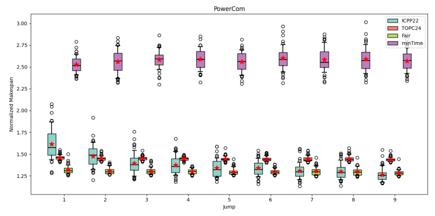
(b) Communication



(c) Amdahl



(d) General



(e) PowerCom

Figure 13: Impact of maximum dependency skip distance (Jump) on scheduling efficiency.

Figures 7 to 13 summarize the detailed experimental results. Each figure corresponds to varying a single parameter listed in Table 7, while keeping all other parameters at their default values. The boxplots depict the distribution of makespan ratios relative to the practical lower bound across multiple instances and heuristics.

D.5 Average Values

Table 8: Average Values for Each Model and Heuristic

Model	ICPP22	TOPC24	Fair	minTime	minArea
Roofline	1.28	1.28	1.28	1.52	11.28
Communication	1.75	1.38	1.38	2.97	10.22
Amdahl	2.32	1.98	2.01	21.31	7.88
General	2.74	1.50	1.60	4.86	5.37
PowerCom	1.52	1.46	1.35	2.84	12.76
Average	1.92	1.52	1.52	6.70	9.50

Table 8 reports the average makespan ratios obtained by each heuristic for each speedup model across all experiments. The last row shows the overall average for each heuristic, facilitating a broad comparison.

D.6 Maximum Values

Table 9 presents the maximum observed makespan ratios for each heuristic-model pair throughout all experimental scenarios. The last row highlights the highest ratio encountered for each heuristic, illustrating worst-case performance.

Table 9: Maximum Values for Each Model and Heuristic

Model	ICPP22	TOPC24	Fair	minTime	minArea
Roofline	2.56	2.56	2.56	3.18	117.75
Communication	2.22	2.17	2.30	8.45	34.30
Amdahl	4.52	3.75	3.76	33.26	23.45
General	5.11	2.64	2.70	10.34	14.76
PowerCom	3.71	2.16	2.46	6.83	69.98
Maximum	5.11	3.75	3.76	33.26	117.75