浙 江 大 學

本 科 生 毕 业 论 文

文献综述和开题报告

姓名与学号	张志心 3210106357
指导教师	谈之奕
年级与专业	2021级数学与应用数学
所在学院	数学科学学院

一、题目：在线带测试多处理器调度问题竞争比下界研究

二、指导教师对文献综述、开题报告、外文翻译的具体要求：

指导教师（签名）_____

年　月　日

目录

一、文献综述

1 背景介绍

1.1 多处理器调度问题

1.1.1 问题的背景与基本概念

多处理器调度问题（Multiprocessor Scheduling Problem）的研究始于计算机系统中对计算资源的有效利用。早期的计算机系统多为单核处理器，任务的执行通常由操作系统进行单核调度。随着多核处理器的普及，计算任务可以在多个处理器上并行执行，从而显著提高计算效率和处理能力。然而，任务之间可能存在依赖关系，或者每个任务的处理时间不同，如何合理地分配任务到各个处理器，最大限度地减少完成时间或优化其他目标，成为了多处理器调度问题的核心。

多处理器调度问题涉及在多个处理器之间合理地分配和调度一组作业或任务。任务之间可能是独立的，也可能存在依赖关系，例如某些任务必须在其他任务完成后才能开始。调度的目标通常包括最小化最大完成时间（Makespan）、最小化任务的平均完成时间、或者最大化系统的吞吐量等。

多处理器调度问题在许多实际应用中都有广泛的应用场景。如：高性能并行计算、实时系统、操作系统需要实时处理多个可并行进行的任务的场景。

我们可以建立这样一个简单的数学模型来描述多处理器调度问题。

定义 1.1 (朴素多处理器调度问题)**.** 一个实例 I 包含一组 n 个作业 $J=\{J_1,J_2,\cdots,J_n\}$，每个作业都要在 m 台并行相同的机器集 $M=\{M_1,M_2,\cdots,M_m\}$ 上非抢占执行。每个作业有一个确定的处理时间上界 p_j。目标是最小化完工时间 C_{max}，即所有任务中最后一个完成的任务的完成时间。

多处理器调度问题有很多变种。例如：最小化任务的完成时间之和、最小化任务的等待时间之和、在限制时间内最大化完成任务的数量等。还有一些变种问题是任务之间存在一些依赖关系，某些任务必须在其他任务完成之后才能开始。而“非抢占执行”的含义是

每个任务只要开始进行就不能中断，与之相对的是“抢占执行”，即任务可以中断并立即切换为另一个任务开始执行。

多处理器调度问题是一个 NPC 问题，即不存在多项式复杂度的确定性求解算法。目前研究者们已经设计出确定性算法如启发式搜索，非确定性算法如遗传算法、模拟退火算法等。

1.1.2 带测试的多处理器调度问题

带测试的多处理器调度问题（Testing Multiprocessor Scheduling Problem）是多处理器调度问题的一个扩展，其中除了通常的任务调度外，还需要进行额外的测试步骤。这些测试步骤用于评估任务的特性（如任务的处理时间、执行顺序等），从而帮助调度器更准确地做出调度决策。与传统的调度问题不同，带测试调度问题需要在优化任务调度的同时考虑测试操作的开销和影响。

带测试的多处理器调度问题一般包含两个重要元素：测试阶段和调度阶段。在测试阶段，系统会对任务进行某些形式的检测或预处理（如任务的处理时间、执行顺序等）。这些测试通常需要消耗时间和计算资源，因此测试的时间需要纳入整个调度优化过程中。

我们可以从上面的实际问题中抽象出一个数学模型：

定义 1.2 (带测试多处理器调度问题)**.** 一个实例 I 包含一组 n 个作业 $J=\{J_1,J_2,\cdots,J_n\}$，每个作业都要在 m 台并行相同的机器集 $M=\{M_1,M_2,\cdots,M_m\}$ 上非抢占执行。每个作业带有一个处理时间上界 u_j 和一个测试操作的时长 t_j，如果进行测试，作业将得到一个实际运行时长 p_j。每一个作业 J_j 都可以选择在某台机器上花 u_j 时间运行，或者先花 t_j 时间进行测试，之后在同一台机器上运行 p_j 时间。目标是最小化完工时间 C_{max}。

当 $t_j=0$ 时，问题就被归约为朴素多处理器调度问题。因此，带测试多处理器调度问题也至少是 NPC 问题，不存在多项式时间复杂度的算法。

1.2 在线、半在线问题和竞争比

1.2.1 在线、半在线带测试多处理器调度问题

在线和半在线多处理器调度问题是多处理器调度领域中的两个重要分支。它们在任务到达时机和调度决策上有所不同，主要涉及如何在信息不完全的情况下，合理调度多个处理器以优化系统性能。随着多核处理器和并行计算的普及，在线和半在线调度问题在实际应用中有着广泛的影响。

定义 1.3 (在线带测试多处理器调度问题)**.** 当一个任务到达时，调度器只能根据当前的信息进行调度，而无法对未来的任务做出预测。因此调度器需要在每个任务到达时就立即决定其处理器分配和执行顺序。

定义 1.4 (半在线带测试多处理器调度问题)**.** 介于在线和离线调度问题之间。在半在线调度中，调度器可以在任务到达之前得知全部的 t_i 和 u_i，但只有在花费 t_i 进行测试之后才可得知 p_i。

与离线算法不同，在线算法无法在一开始就拥有全部输入数据，而是必须随着输入的逐步到来，作出即时决策。由于无法预知未来的输入数据，在线算法通常面临一个重要问题：如何在信息不完全的情况下做出近似最优的决策。

1.2.2 竞争比

为了衡量在线算法的表现，竞争比（Competitive Ratio）是一个关键的概念。它用于比较在线算法的性能与最佳离线算法（或最优算法）在同样问题中的表现差异。

定义 1.5. 在线算法的竞争比是一个用于度量其性能的度量值。它定义为在线算法执行的最坏情况性能与离线算法执行的最优性能之间的比值。具体来说，假设有一个在线调度问题，其实例为 I，并记所有实例的集合为 $\mathscr{I}$，定义

- $C(I)$ 是某个离线算法的最优解，即假设我们能够提前知道所有任务的输入，可以通过全局优化来得到的最优解。
- $C^*(I)$ 是在线算法在最坏情况下的解。

则在线算法的竞争比 α 可以表示为

$$\alpha = \sup_{I \in \mathscr{I}} \frac{C(I)}{C^*(I)}$$

竞争比总是不小于 1 的，其实际意义在于：

1. 衡量在线算法的优劣：竞争比为评估在线算法的性能提供了一个标准。它告诉我们，在最坏的情况下，在线算法的表现相比于最优解有多差。例如，某些算法的竞争比是常数（如 3-竞争），这意味着无论输入数据如何变化，算法的执行时间或完成度始终不会超过最优解的 3 倍。

2. 适应性与实时性：在线算法的竞争比越低，说明它能在实时情况下做出更接近最优的决策。例如，在云计算和数据流处理等实时计算中，能够提供一个低竞争比的在线调度算法，对于高效利用计算资源、减少延迟至关重要。

3. 算法的鲁棒性：在面对不同类型的输入时，竞争比高的算法可能在某些场景下表现不佳。通过研究竞争比，我们能够了解在不同场景下算法的鲁棒性，进而选择更适合特定应用的算法。

1.2.3 竞争比下界

在线算法的竞争比下界是指在特定问题中，任何在线算法在最坏情况下的性能相对于最优解的比例所能达到的最小值。了解这一下界具有重要的理论意义，因为它为我们提供了一个不可逾越的性能界限，帮助我们理解在线算法的最佳表现。具体来说，如果一个问题的竞争比下界是 α_0，则意味着无论我们如何设计在线算法，算法在最差情况下的表现都至少是最优解的 α 倍。

研究一个问题的竞争比下界有时是比研究它的具体算法更有效的。具体来说，它可以

1. 揭示一个问题在最坏情况下的不可逾越的性能界限。

2. 帮助评估算法的实际应用性能：竞争比下界表示了在线算法在最坏情况中的最大劣化程度，而通过与实际算法的对比，我们可以判断一个在线算法是否能在实际应用中达到一个“可接受”的水平。

3. 为算法设计提供指导：了解竞争比下界有助于为在线算法的设计提供理论依据。如果某个问题的竞争比下界较高，那么即使在设计过程中对算法进行大量优化，期望算法的性能会接近最优解也是不现实的。相反，如果下界较低，那么设计一个接近最优的在线算法是可能的。

2 国内外研究现状

我们在本文中研究完全在线的多处理器调度与测试问题[1-4]，并从随机化算法的角度进行研究。多处理器调度[5]是最早的已知 NP 难组合优化问题之一，在过去几十年中得到了广泛研究。多处理器调度的一个实例 I 包含一组 n 个作业 $J=\{J_1,J_2,\ldots,J_n\}$，每个作业都要在一组 m 台并行的相同机器 $M=\{M_1,M_2,\ldots,M_m\}$ 上非抢占地执行，目标是最小化最大完工时间 $C_{\max}$，即最大作业完成时间。与经典设置不同的是，在调度与测试中，每个作业 J_j 都有一个处理时间上界 u_j，并且有一个长度为 t_j 的测试操作，但其处理时间 p_j 直到作业被测试后才会揭示。作业 J_j 可以在其中一台机器上执行 u_j 时间，或者选择先进行测试，测试时间为 t_j，然后立即执行 p_j 时间。若所有作业都在时间零到达，则多处理器调度与测试是一个半在线问题，表示为 $P|t_j,0\le p_j\le u_j|C_{\max}$。本文研究的是完全在线问题，其中作业按顺序到达，在作业到达时需要做出测试决策，并且指定测试和/或执行的机器，表示为 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$。显然，半在线是完全在线的特例，在这两种情况下，调度器都应该利用已知的作业信息，在作业到达时决定是否进行测试，以便最好地平衡由于未知处理时间所消耗的总时间。

给定一个多项式时间的确定性算法，针对半在线或完全在线问题，令 $C(I)$ 为该算法在实例 I 上产生的最大完工时间，而 $C^*(I)$ 为最优离线调度的最大完工时间。算法的性能通过竞争比来衡量，定义为 $\sup_I\{C(I)/C^*(I)\}$，其中 I 遍历所有问题实例，算法称为 $\sup_I\{C(I)/C^*(I)\}$-竞争的。切换到随机化算法时，我们相应地收集其在实例 I 上的期望最大完工时间 $E[C(I)]$，该随机化算法称为 $\sup_I\{E[C(I)]/C^*(I)\}$-竞争的。对于在线问题，随机化算法有时可以更好地处理不确定性，从而导致比最好的确定性算法更低的期望竞争比。我们为 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 提出了这样的一个随机化算法，并且进一步证明，当只有两台机器时，其期望竞争比严格小于任何确定性算法已证明的竞争比下界。在我们的问题中，作业处理是非抢占的。

在文献中，研究人员还考虑了抢占式作业处理[1-4]，其中任何测试或执行操作都可以被中断并稍后恢复，或者考虑了更为受限的测试抢占式变体[4]，在该变体中，已测试作业的测试和执行操作是非抢占的，但执行操作不必立即跟随测试操作，且不一定发生在同一台机器上。此外，我们的目标是最小化最大完工时间 $C_{\max}$，即最小最大目标；而另一个重要目标是最小化总作业完成时间，或者最小和目标，这也是受到了许多研究的关注[1-2,4]。

2.1 全在线问题研究进展

现有的针对完全在线问题 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 的近似算法，无论是确定性算法还是随机化算法，都不多。我们首先区分一个特殊的情况，其中所有的测试操作时间均为单位时间，即对于每个作业 J_j，有 $t_j=1$，我们称之为均匀测试情况[1-2,4]，表示为 $P|\text{online},t_j=1,0\le p_j\le u_j|C_{\max}$。注意，在一般的测试情况下，测试时间可以是任何非负值。当只有一台机器时，机器上的作业处理顺序与最大完工时间无关。这表明完全在线问题和半在线问题是相同的。第一组结果是关于半在线均匀测试问题 $P_1|t_j=1,0\le p_j\le u_j|C_{\max}$，由 D¨urr 等人[1-2]提出。他们提出，当 $u_j\ge\varphi=\frac{\sqrt{5}+1}{2}$ 时测试作业 J_j，或者以概率 $f(u_j)=\max\left(0,\frac{u_j(u_j-1)}{u_j(u_j-1)+1}\right)$ 测试它，从而得到了一个确定性的 φ-竞争算法和一个期望的 4/3 竞争算法。令 $r_j=\frac{u_j}{t_j}$；Albers 和 Eckl[3]将上述两种算法扩展到一般测试情况下 $P_1|t_j,0\le p_j\le u_j|C_{\max}$，即当 $r_j\ge\varphi$ 时测试作业 J_j，或者以概率 $f(r_j)$ 测试它，分别达到了相同的竞争比和期望竞争比。作者们[1-3]证明了这两种算法是最优的，即 φ 是任何确定性算法的竞争比下界，而 4/3 是任何随机化算法的期望竞争比下界，这一结果由 Yao 原理[6]得出。

当至少有两台机器时，完全在线问题比半在线问题更为一般。Albers 和 Eckl[4]提出了在列表调度规则[7]中测试作业 J_j，即当 $r_j\ge\varphi$ 时，按照该规则将每个作业分配到最空闲的机器上进行处理（可能是测试后执行）。他们证明了这样的算法是 $\varphi(2-\frac{1}{m})$-竞争的，且该分析是紧的，其中 m 是机器的数量。他们还证明了，即使在均匀测试情况下 $P|\text{online},t_j=1,0\le p_j\le u_j|C_{\max}$ 中，任何确定性算法的竞争比下界为 2[4]，并且对于两台机器的一般测试情况 $P_2|\text{online},t_j,0\le p_j\le u_j|C_{\max}$，即当只有两台机器时，其下界为 2.0953，这一结果也稍微更好[4]。

2.2 半在线问题研究进展

当只有一个机器时，半在线问题与全在线问题等价。

当至少有两台机器时，因为所有作业都在时间零到达，因此调度器可以利用所有已知的 u_j 和 t_j 值，不仅用于作业测试决策，还可以形成某些作业处理顺序，以更好地减少最大完工时间。事实上，这正是 Albers 和 Eckl[4] 在一般测试情况下提出的开创性所谓 SBS 算法的情况，该算法在机器数量趋向无穷大时为 3.1016-竞争算法。对于均匀测试情况，Albers 和 Eckl[4] 还提出了一个 3-竞争算法，并给出了对于任何确定性算法的竞争比下界 $\max\{\varphi, 2-\frac{1}{m}\}$。

上述两个竞争比在一般和均匀测试情况下分别由 Gong 和 Lin[8] 改进为 2.9513 和 2.8081，最新的竞争比是 Gong 等人[9] 分别提出的 2.8019 和 2.5276。

为了证明竞争比下界 $2-\frac{1}{m}$，Albers 和 Eckl[4] 提出了一个 $P|t_j=1, 0\le p_j\le u_j|C_{\max}$ 的实例，迫使任何确定性算法对所有作业进行测试。这意味着该下界不仅适用于任何确定性算法的竞争比，也适用于任何随机化算法的期望竞争比，依据 Yao 原理[6]。此外，$2-\frac{1}{m}$ 的下界因此适用于完全在线均匀测试问题 $P|\text{online}, t_j=1, 0\le p_j\le u_j|C_{\max}$ 中任何随机化算法的期望竞争比。

2.3 其他相关工作

Dürr 等人[1-2] 为单机均匀测试问题 $P_1|t_j=1, 0\le p_j\le u_j|\sum_j C_j$ 提出了一个期望竞争比为 1.7453 的随机化算法，该算法按均匀随机顺序测试待测作业。他们还证明了期望竞争比的下界为 1.6257。Albers 和 Eckl[3] 为一般测试情况 $P_1|t_j, 0\le p_j\le u_j|\sum_j C_j$ 设计了一个随机化算法，其中作业 J_j 根据比率 r_j 使用一个相当复杂的概率函数进行测试；该算法的期望竞争比为 3.3794。

回顾经典在线多处理器调度问题 $P|\text{online}|C_{\max}$ 的最先进的随机化算法，其中作业依次到达，处理时间 p_j 在作业 J_j 到达时揭示，并且在到达时必须无条件地分配作业到某台机器上执行，以最小化最大完工时间。众所周知，对于这个问题，从第一个确定性算法列表调度[7] 的竞争比 2 到当前最好的 1.9201[10]，经历了很长时间的改进。对于每个在线近似算法，性能分析中最具挑战性的一部分是精确估计最优离线最大完工时间。三个最典型的下

界是 $\frac{1}{m}\sum_{j=1}^{n} p_j$（平均机器负载）、$\max_{j=1}^{n} p_j$（最大的作业处理时间）和 $p[m]+p[m+1]$，其中 $p[j]$ 是所有作业中的第 j 大处理时间。Albers[11] 证明，如果只使用上述三个下界，则任何确定性算法的竞争比在 m 趋于无穷大时都不能小于 1.919。

Albers[11] 提出了一个期望竞争比为 1.916 的随机化算法，并且在性能分析中仅使用了上述三个下界来估计最优离线最大完工时间。更详细地说，Albers 提出了两个确定性算法 A_0 和 A_1，并提议以 0.5 的概率执行每个算法。作者表明，如果期望最大完工时间与这三个下界相比过大，即 A_0 和 A_1 都产生了较大的最大完工时间，那么在实例中会有许多大作业，使得最优离线最大完工时间无法太小。

这种由常数数量的确定性算法组成的随机化算法被称为“几乎随机化算法”[12]，它利用其优秀的组件算法，在至少一个组件算法表现良好时，它能够表现良好，而如果没有任何组件算法表现良好，那么最优离线最大完工时间也会远离下界。这个设计思想已被用于逼近许多其他优化问题，并取得了突破性的成果，例如其他调度问题[9,13] 和 k-服务器问题[14]。事实上，对于在线多处理器调度问题 $P|\text{online}|C_{\max}$，在[11] 之前，Seiden[13] 修改了 Bartal 等人[15] 和 Seiden[16] 的非几乎随机化算法，这些算法将当前作业分配给两台负载最小的机器之一，并以一定的概率运行，变成了几乎随机化算法，这是一个 k 个确定性算法的均匀分布，其中 k 是算法内要创建的调度的数量，并且选择足够大以保证期望竞争比。尽管这种方法对小 m（特别是 $m \leq 7$）有效，Albers[11] 观察到[13,15-16] 中算法的分析方法在一般的大 m 情况下不起作用，随后提出了基于仅两个确定性算法的新几乎随机化算法。

2.4 存在问题

现有的针对全在线问题的竞争比下界的研究主要还停留在特殊情况（$m=1$）和小范围情况（$m \leq 2, n \leq 3$）。研究方法也局限在人工构造博弈决策树，当决策树的层数更深（对应 n 更大）或点数更多（对应 m 更大）时，人工构造就已经很难奏效。事实上，对于 $m \geq 3, n \geq 4$ 的情形，现有的研究都只能仿照 $m=2, n=3$ 的构造方式尝试构造较坏情况的数据，这导致目前的竞争比下界和竞争比上界（即目前已经存在的在线算法的竞争比）仍有较大差距。

3 研究展望

由上可见，竞争比下界仍可能有较大的提升空间。因此设计一个通用的建立博弈搜索树的算法是有必要的。借助计算机的强大算力，我们可以扩大搜索范围，发现人力无法观察到的极端数据，进一步提升竞争比下界。

4 参考文献

[1] DÜRR C, ERLEBACH T, MEGOW N, et al. Scheduling with explorable uncertainty[C]//9th Innovations in Theoretical Computer Science Conference (ITCS 2018). 2018.

[2] DÜRR C, ERLEBACH T, MEGOW N, et al. An adversarial model for scheduling with testing[J]. Algorithmica, 2020, 82(12): 3630-3675.

[3] ALBERS S, ECKL A. Explorable uncertainty in scheduling with non-uniform testing times[C]//Approximation and Online Algorithms: 18th International Workshop, WAOA 2020, Virtual Event, September 9–10, 2020, Revised Selected Papers 18. 2021: 127-142.

[4] ALBERS S, ECKL A. Scheduling with testing on multiple identical parallel machines[C]//Algorithms and Data Structures: 17th International Symposium, WADS 2021, Virtual Event, August 9–11, 2021, Proceedings 17. 2021: 29-42.

[5] GAREY MICHAEL R, JOHNSON DAVID S. Computers and Intractability: A guide to the theory of NP-completeness[Z]. 1979.

[6] YAO A C C. Probabilistic computations: Toward a unified measure of complexity[C]//18th Annual Symposium on Foundations of Computer Science (sfcs 1977). 1977: 222-227.

[7] GRAHAM R L. Bounds for certain multiprocessing anomalies[J]. Bell system technical journal, 1966, 45(9): 1563-1581.

[8] GONG M, LIN G. Improved approximation algorithms for multiprocessor scheduling with testing[C]// International Workshop on Frontiers in Algorithmics. 2021: 65-77.

[9] GONG M, CHEN Z Z, LIN G, et al. Randomized algorithms for fully online multiprocessor scheduling with testing[J]. ArXiv preprint arXiv:2305.01605, 2023.

[10] FLEISCHER R, WAHL M. On-line scheduling revisited[J]. Journal of Scheduling, 2000, 3(6): 343-353.

[11] ALBERS S. On randomized online scheduling[C]//Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. 2002: 134-143.

[12] REINGOLD N, WESTBROOK J, SLEATOR D D. Randomized competitive algorithms for the list update problem[J]. Algorithmica, 1994, 11(1): 15-32.

[13] SEIDEN S. Barely random algorithms for multiprocessor scheduling[J]. Journal of Scheduling, 2003, 6(3): 309-334.

[14] BARTAL Y, CHROBAK M, LARMORE L L. A randomized algorithm for two servers on the line[J]. Information and Computation, 2000, 158(1): 53-69.

[15] BARTAL Y, FIAT A, KARLOFF H, et al. New algorithms for an ancient scheduling problem[C]// Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. 1992: 51-58.

[16] SEIDEN S S. Online randomized multiprocessor scheduling[J]. Algorithmica, 2000, 28(2): 173-216.

二、开题报告

1 问题提出的背景

1.1 背景介绍

1.1.1 多处理器调度问题

多处理器调度问题（Parallel Machine Scheduling Problem）[1] 是生产调度领域中的一个经典问题，它主要关注如何在 m 台相同的机器上合理安排 n 个任务的执行顺序，以达到某些特定的目标，比如最小化所有任务完成的时间总和、最小化最后一个任务的完成时间等。形式化地，这个问题可以描述为：

定义 1.1. 朴素多处理器调度问题：一个实例 I 包含一组 n 个作业 $J=\{J_1,J_2,\cdots,J_n\}$，每个作业都要在 m 台并行相同的机器集 $M=\{M_1,M_2,\cdots,M_m\}$ 上非抢占执行。每个作业有一个确定的处理时间上界 p_j。目标是最小化完工时间 C_{max}，即所有任务中最后一个完成的任务的完成时间。

然而在实际问题中，作业的实际完工时间可能需要测试才能得出。由此可以抽象出多处理器调度问题的一个变体[2][3][4]：

定义 1.2. 带测试多处理器调度问题：一个实例 I 包含一组 n 个作业 $J=\{J_1,J_2,\cdots,J_n\}$，每个作业都要在 m 台并行相同的机器集 $M=\{M_1,M_2,\cdots,M_m\}$ 上非抢占执行。每个作业带有一个处理时间上界 u_j 和一个测试操作的时长 t_j，如果进行测试，作业将得到一个实际运行时长 p_j。每一个作业 J_j 都可以选择在某台机器上花 u_j 时间运行，或者先花 t_j 时间进行测试，之后在同一台机器上运行 p_j 时间。目标是最小化完工时间 C_{max}。

这类问题广泛存在于制造业、云计算资源分配、物流运输等多个行业背景中。例如，在制造业中，工厂需要决定将哪些工件分配给哪几台机器进行加工以及加工的先后顺序；在云计算环境中，需要决定如何高效地为虚拟机分配计算资源。解决多处理器调度问题对于提高生产效率、降低成本具有重要意义。针对不同场景下的具体需求，研究者们提出了

多种算法来寻找最优解或近似解，有确定性算法例如启发式算法；非确定性算法（随机算法）例如遗传算法、模拟退火算法等。

1.1.2 在线问题和竞争比

在实际问题中，我们往往不能在一开始就得到所有数据，而只能随着决策的过程逐步获取数据。这就引出了离线问题、半在线问题和在线问题的概念。

定义 1.3. 离线带测试多处理器调度问题：所有作业在时间零时刻到达，且 t_j, p_j, u_j 全部事先已知。此时问题等价于处理时间上界为 $\min(t_j + p_j, u_j)$ 的朴素多处理器调度问题。

定义 1.4. 半在线带测试多处理器调度问题：所有作业在时间零时刻到达，记作 $P \mid t_j, 0 \leq p_j \leq u_j \mid C_{max}$。

定义 1.5. 在线带测试多处理器调度问题：调度器需要在作业到达时做出测试决策，并指定机器进行测试或直接执行，每个作业在上一个作业的调度决策完成后到达，记作 $P \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{max}$。

由于信息的缺失，在一般情况下我们无法设计出能够确定地作出最优决策的算法。因此只能退而寻求近似算法。我们用竞争比的概念来描述算法的性能：

定义 1.6. 确定性算法的竞争比：假设存在一个确定性算法，令 $C(I)$ 表示该算法在实例 I 上产生的完工时间，$C^*(I)$ 表示最优离线调度的完工时间。定义该算法的竞争比为

$$\alpha = \sup_I \frac{C(I)}{C^*(I)}, \quad \text{其中} I \text{遍历所有问题实例},$$

该算法被称为 α-竞争算法。

提升竞争比下界对于设计在线和半在线算法具有重要的研究意义：它可以为在线算法提供一个性能保障，即使在最不利的情况下，也能保证算法的表现不会太差；它还有助于指导算法的设计和改进，如果一个在线算法的竞争比已经接近或达到已知的下界，那么进一步优化的空间可能非常有限。

1.1.3 竞争比下界的现有成果

定理 1.7. 对于问题 $P2\,|\,online,t_j,0\le p_j\le u_j\,|\,C_{max}$，任何确定性算法的竞争比大于 *2.2117*。[4]

证明. 一个可行的构造方案如下：定义 $\varphi=\frac{\sqrt{5}+1}{2}$（事实上，可以定义 φ 为 (1.5,2) 的任意实数。）令 $t_i=1,\forall i$。有两台机器，令其为 M_1,M_2，三个工件为 J_1,J_2,J_3。令 $u_1=\varphi$，将 J_1 在 M_1 上加工。

Case1: 若对 J_1 测试，此时 $p_1=\varphi$，令 $u_2=\varphi$。

Case1.1: 若对 J_2 测试，此时 $p_2=\varphi$。如果将 J_2 放在 M_1 上加工，此时 $C=2+2\varphi$，而 $C^*=\varphi$，$C/C^*=2\varphi+2/\varphi>2$。如果将 J_2 放在 M_2 上加工，令 $u_3=2\varphi$，若 J_3 测试，此时 $p_3=2\varphi$，则此时 $C\ge 2+3\varphi$，而 $C^*=2\varphi$，$C/C^*=2+3\varphi/2\varphi>2$。若 J_3 不测试，此时 $p_3=0$，则此时 $C\ge 1+3\varphi$，而 $C^*=1+\varphi$，$C/C^*=1+3\varphi/1+\varphi>2$。

Case1.2: 若对 J_2 不测试，此时 $p_2=0$。如果将 J_2 放在 M_1 上加工，此时 $C=2\varphi+1$，而 $C^*=\varphi$，$C/C^*=2\varphi+1/\varphi>2$。如果将 J_2 放在 M_2 上加工，令 $u_3=1+\varphi$，若 J_3 测试，此时 $p_3=1+\varphi$，则此时 $C\ge 2+2\varphi$，而 $C^*=1+\varphi$，$C/C^*=2+2\varphi/1+\varphi=2$。若 J_3 不测试，此时 $p_3=0$，则此时 $C\ge 1+2\varphi$，而 $C^*=2$，$C/C^*=1+2\varphi/2>2$。

Case2: 若不对 J_1 测试，此时 $p_1=0$，令 $u_2=\varphi$。

Case2.1: 若对 J_2 测试，此时 $p_2=\varphi$。如果将 J_2 放在 M_1 上加工，此时 $C=1+2\varphi$，而 $C^*=\varphi$，$C/C^*=2\varphi+1/\varphi>2$。如果将 J_2 放在 M_2 上加工，令 $u_3=1+\varphi$，若 J_3 测试，此时 $p_3=1+\varphi$，则此时 $C\ge 2+2\varphi$，而 $C^*=1+\varphi$，$C/C^*=2+2\varphi/1+\varphi=2$。若 J_3 不测试，此时 $p_3=0$，则此时 $C\ge 1+2\varphi$，而 $C^*=2$，$C/C^*=1+2\varphi/2>2$。

Case2.2: 若对 J_2 不测试，此时 $p_2=0$。如果将 J_2 放在 M_1 上加工，此时 $C=2\varphi$，而 $C^*=1$，$C/C^*=2\varphi/1>2$。如果将 J_2 放在 M_2 上加工，令 $u_3=1+\varphi$，若 J_3 测试，此时 $p_3=1+\varphi$，则此时 $C\ge 2+2\varphi$，而 $C^*=1+\varphi$，$C/C^*=2+2\varphi/1+\varphi=2$。若 J_3 不测试，此时 $p_3=0$，则此时 $C\ge 1+2\varphi$，而 $C^*=2$，$C/C^*=1+2\varphi/2>2$。

□

对于所有测试操作时间都为单位时间的问题 $P\,|\,\text{online},t_j=1,0\le p_j\le u_j\,|\,C_{\max}$：当只有一台机器时，Dühr 等[2][3]提出，当 $u_j\le\varphi:=\frac{\sqrt{5}+1}{2}$ 时，对作业 j 进行测试，可以得到 φ-竞争算法；或者以概率 $f(u_j)=\max\left(0,\frac{u_j(u_j-1)}{u_j(u_j-1)+1}\right)$ 概率对作业进行测试，可以得到 $\frac{4}{3}$-竞争的

随机化算法。论文[5]，这两种算法是最优的，即 φ 是确定性算法的竞争比下界，$\frac{4}{3}$ 是任何随机化算法的期望竞争比下界。当有 $m \geq 2$ 台机器时，存在 $\varphi(2-\frac{1}{m})$-竞争算法[6]。论文[4]中提到了，对于任何确定性算法，竞争比下界为 2。该下界的证明使用了两台机器、三个工件来构造最坏实例，具体构造方法可以参考定理1.7 的证明。另外，在 Gong 的论文[7]中，进一步证明了任何确定性算法的竞争比下界为 2.2117；同时还提出了一种确定性算法集成的随机化算法，其期望竞争比为 $\frac{3\varphi+3\sqrt{13-7\varphi}}{4} \approx 2.1839$。

1.2 本研究的意义和目的

多处理器调度问题作为一个重要的组合优化问题，其在实际生产环境中具有广泛的应用场景，如工业生产和资源分配等领域。带测试的多处理器调度问题引入了相同的工作的不同处理方式（一种未知效果的新处理手段），并以提高整体系统的效率为目标，同样有着重要的研究意义。

目前在线带测试多处理器调度问题竞争比下界还有较大的提升空间。现有的研究主要集中在机器数量、工作数量较少的情况下，使用手动构造最坏情况来分析竞争比下界，但对于更一般的情形，特别是当 $m \geq 3, n \geq 4$ 时，现有的竞争比下界仍有改进余地。

通过引入计算机辅助方法，利用程序设计方法更高效地构建和分析决策树，可以扩大搜索范围，有希望进一步提高竞争比下界。

2 项目的主要内容和技术路线

2.1 主要研究内容

本研究旨在探索并证明在线带测试的多处理器调度问题的确定性算法竞争比下界。即：证明不存在竞争比总是低于某个下界的确定性算法。

我将对问题进行程序建模，通过分析特定情况下的竞争比表现，识别出影响竞争比的关键因素。在此基础上，我还将对现有算例进行分析，特别关注那些导致竞争比达到最坏的特殊情况。最后，我将运用计算机辅助的方法在有限的时间内搜索出更广泛的实例，以验证该问题的竞争比下界，并通过严谨的分析来确认最终结果。

2.2 技术路线

我将尝试沿用 Gormley 等设计的线性在线问题的竞争比下界求解模型[8]，将其应用到在线带测试的多处理器调度问题的竞争比下界的研究中。依据该论文的思路，我将先建立传统的博弈搜索模型，将调度的过程视为调度算法与对手的 n 轮博弈过程。每一轮博弈分为以下三步：第一步对手指定当前任务的 t_i 和 u_i；第二步算法为当前输入的任务分配一个处理器并决定是否测试; 第三步若算法决定测试则对手还要指定当前任务的 p_i。算法的最终目的是达到预设的竞争比 α，而对手的最终目的则是阻止算法达到该竞争比。如果最终的博弈结果是对手获胜，则证明无论如何决策，总存在一个实例使竞争比大于 α，即竞争比下界大于 α。

传统的博弈搜索模型将使用搜索剪枝的方法，暴力搜索所有可能的数据并剪去目前已不可能达到最优解的数据。Gormley 的模型[8]则考虑到了调度问题的“线性性”，即博弈搜索树的构建可视为一个带 max、min 运算的线性规划模型，进而通过拆分 max 和 min 运算得到多个传统线性规划模型。从而一方面可以直接利用线性规划算法求解，另一方面也可以通过线性规划的解的性质，将解的范围限制在有限的空间内，对对手的决策进行优化来辅助剪枝。

最后，我将用 C++ 语言实现上述最优化模型和搜索框架，先对 $m=1$ 和 $m=2$ 的具有较为成熟的情形验证框架的准确性，再探索 $m=3, n\geq 4$ 更精确的竞争比下界。

2.3 可行性分析

现有的对竞争比下界的研究大多基于手工构造特殊数据。这对 $m\geq 3, n\geq 4$ 的情况已经很难奏效。因此借助计算机的强大算力构造人力难以搜索到的数据来提升竞争比下界是大有可为的。

此外，Gormley[8] 的论文中引用了一个定理：

定理 2.1. 若在线问题是线性的，给定任意实数 $\alpha>1$ 和整数 n，存在一个算法，可在有限时间内构造出 n 层的博弈树证明在线问题的竞争比下界不小于 α，或证明竞争比下界大于 α。

这个定理保证了算法的可停机性。然而，由于直接拆解最优化问题会导致子问题的指

数级增加，我们仍需设计一些搜索剪枝的操作以提高程序上运行效率，使其能在有限的时间内寻找到具有更大竞争比下界的实例。

3 研究计划进度安排及预期目标

3.1 进度安排

1. 2025.2.28 以前，完成问题建模和文献调研，归纳现有方法。完成三合一等准备工作。
2. 2025.3.15 以前，设计计算机搜索的代码并初步验证现有研究成果。
3. 2025.3.31 以前，通过进一步的理论分析，确定搜索剪枝策略。
4. 2025.4.15 以前对剪枝策略和现有运行结果，进行深入理论分析和证明，进一步改进竞争比下界。
5. 2025.5 以前，撰写毕业论文，持续尝试改进搜索策略，以得到更好的结果和获得更快的程序验证效率。

3.2 预期目标

我希望能够深入分析多处理器调度问题的数学模型，基于对于构建决策者和问题之间的博弈形成的决策树模型，探索在调度问题中形成的算法策略，发掘出竞争比低的在线算法。同时，通过使用数学工具和计算机辅助的方式，进行广泛的实例模拟，以验证现有研究成果中的竞争比下界。同时，我希望在原有的研究成果基础上，通过计算机进行高效的剪枝搜索，进一步提高机器数量 ≥ 3，工件数量 ≥ 4 的期望竞争比的下界。

4 参考文献

[1] GARY M R, JOHNSON D S. Computers and Intractability: A Guide to the Theory of NP-completeness[Z]. 1979.

[2] DÜRR C, ERLEBACH T, MEGOW N, et al. Scheduling with explorable uncertainty[C]//9th Innovations in Theoretical Computer Science Conference (ITCS 2018). 2018.

[3] DÜRR C, ERLEBACH T, MEGOW N, et al. An adversarial model for scheduling with testing[J]. Algorithmica, 2020, 82(12): 3630-3675.

[4] ALBERS S, ECKL A. Scheduling with testing on multiple identical parallel machines[C]//Algorithms and Data Structures: 17th International Symposium, WADS 2021, Virtual Event, August 9–11, 2021, Proceedings 17. 2021: 29-42.

[5] ALBERS S, ECKL A. Explorable uncertainty in scheduling with non-uniform testing times[C]//Approximation and Online Algorithms: 18th International Workshop, WAOA 2020, Virtual Event, September 9–10, 2020, Revised Selected Papers 18. 2021: 127-142.

[6] GRAHAM R L. Bounds for certain multiprocessing anomalies[J]. Bell system technical journal, 1966, 45(9): 1563-1581.

[7] GONG M, CHEN Z Z, LIN G, et al. Randomized algorithms for fully online multiprocessor scheduling with testing[J]. ArXiv preprint arXiv:2305.01605, 2023.

[8] GORMLEY T, REINGOLD N, TORNG E, et al. Generating adversaries for request-answer games[C]// Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms. 2000: 564-565.

三、外文翻译: 在线带测试多处理器调度问题的随机算法

论文标题：Randomized algorithms for fully online multiprocessor scheduling with testing

作者：Mingyang Gong 等

摘要

我们贡献了第一个随机化算法，该算法是任意多个确定性算法在完全在线多处理器调度与测试问题中的集成。当只有两个机器时，我们展示了通过两个组件算法，其期望竞争比已经严格小于已证明的最佳确定性竞争比下界。这种算法结果在文献中极为罕见。多处理器调度是最早得到大量研究的组合优化问题之一。最近，多个研究小组研究了其测试变体，在该变体中，每个作业 J_j 具有一个处理时间的上界 u_j 和一个长度为 t_j 的测试操作；可以选择执行作业 J_j 的最大处理时间 u_j，或者选择对 J_j 进行测试，测试时间为 t_j，以获得确切的处理时间 p_j，然后立即执行该作业的 p_j 时间。我们的目标问题是完全在线多处理器调度与测试问题，其中作业依次到达，因此需要在作业到达时做出测试决策以及分配指定的机器。我们首先利用 Yao 原理证明了至少三台机器情况下和仅两台机器情况下随机算法的期望竞争比下界分别为 1.6682 和 1.6522，并提出了一个期望竞争比为 $(\sqrt{\varphi'+3}+1)(\approx 3.1490)$-竞争的随机算法，该算法是通过一个非均匀概率分布集成了任意多个确定性算法，其中 $\varphi'=\frac{\sqrt{5}+1}{2}$ 是黄金比例。当只有两台机器时，我们展示了基于两个确定性算法的随机化算法已经期望地 $\frac{3\varphi+3\sqrt{13-7\varphi}}{4}(\approx 2.1839)$-竞争，同时证明了任何确定性算法的竞争比下界为 2.2117。

关键词： 调度问题；多处理器；带测试调度；完工时间；随机算法

1 介绍

我们在本文中研究完全在线的多处理器调度与测试问题[1-4]，并从随机化算法的角度进行研究。多处理器调度[5]是最早的已知 NP 难组合优化问题之一，在过去几十年中得到了广泛研究。多处理器调度的一个实例 I 包含一组 n 个作业 $J=\{J_1,J_2,\ldots,J_n\}$，每个作业都要在一组 m 台并行的相同机器 $M=\{M_1,M_2,\ldots,M_m\}$ 上非抢占地执行，目标是最小化最大完工时间 $C_{\max}$，即最大作业完成时间。与经典设置不同的是，在调度与测试中，每个作业 J_j 都有一个处理时间上界 u_j，并且有一个长度为 t_j 的测试操作，但其处理时间 p_j 直到作业被测试后才会揭示。作业 J_j 可以在其中一台机器上执行 u_j 时间，或者选择先进行测试，测试时间为 t_j，然后立即执行 p_j 时间。若所有作业都在时间零到达，则多处理器调度与测试是一个半在线问题，表示为 $P|t_j,0\le p_j\le u_j|C_{\max}$。本文研究的是完全在线问题，其

中作业按顺序到达，在作业到达时需要做出测试决策，并且指定测试和/或执行的机器，表示为 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$。显然，半在线是完全在线的特例，在这两种情况下，调度器都应该利用已知的作业信息，在作业到达时决定是否进行测试，以便最好地平衡由于未知处理时间所消耗的总时间。

给定一个多项式时间的确定性算法，针对半在线或完全在线问题，令 $C(I)$ 为该算法在实例 I 上产生的最大完工时间，而 $C^*(I)$ 为最优离线调度的最大完工时间。算法的性能通过竞争比来衡量，定义为 $\sup_I\{C(I)/C^*(I)\}$，其中 I 遍历所有问题实例，算法称为 $\sup_I\{C(I)/C^*(I)\}$-竞争的。切换到随机化算法时，我们相应地收集其在实例 I 上的期望最大完工时间 $E[C(I)]$，该随机化算法称为 $\sup_I\{E[C(I)]/C^*(I)\}$-竞争的。对于在线问题，随机化算法有时可以更好地处理不确定性，从而导致比最好的确定性算法更低的期望竞争比。我们为 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 提出了这样的一个随机化算法，并且进一步证明，当只有两台机器时，其期望竞争比严格小于任何确定性算法已证明的竞争比下界。在我们的问题中，作业处理是非抢占的。

在文献中，研究人员还考虑了抢占式作业处理[1-4]，其中任何测试或执行操作都可以被中断并稍后恢复，或者考虑了更为受限的测试抢占式变体[4]，在该变体中，已测试作业的测试和执行操作是非抢占的，但执行操作不必立即跟随测试操作，且不一定发生在同一台机器上。此外，我们的目标是最小化最大完工时间 $C_{\max}$，即最小最大目标；而另一个重要目标是最小化总作业完成时间，或者最小和目标，这也是受到了许多研究的关注[1-2,4]。

1.1 关于全在线问题的现有研究

现有的针对完全在线问题 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 的近似算法，无论是确定性算法还是随机化算法，都不多。我们首先区分一个特殊的情况，其中所有的测试操作时间均为单位时间，即对于每个作业 J_j，有 $t_j=1$，我们称之为均匀测试情况[1-2,4]，表示为 $P|\text{online},t_j=1,0\le p_j\le u_j|C_{\max}$。注意，在一般的测试情况下，测试时间可以是任何非负值。当只有一台机器时，机器上的作业处理顺序与最大完工时间无关。这表明完全在线问题和半在线问题是相同的。第一组结果是关于半在线均匀测试问题 $P_1|t_j=1,0\le p_j\le u_j|C_{\max}$，由 D¨urr 等人[1-2]提出。他们提出，当 $u_j\ge\varphi=\frac{\sqrt{5}+1}{2}$ 时测试作业 J_j，或者以概率 $f(u_j)=\max\left(0,\frac{u_j(u_j-1)}{u_j(u_j-1)+1}\right)$ 测试它，从而得到了一个确定性的 φ-竞争算法和一个期望的 4/3 竞争算

法。令 $r_j=\frac{u_j}{t_j}$；Albers 和 Eckl[3]将上述两种算法扩展到一般测试情况下 $P_1|t_j,0\le p_j\le u_j|C_{\max}$，即当 $r_j\ge\varphi$ 时测试作业 J_j，或者以概率 $f(r_j)$ 测试它，分别达到了相同的竞争比和期望竞争比。作者们[1-3]证明了这两种算法是最优的，即 φ 是任何确定性算法的竞争比下界，而 4/3 是任何随机化算法的期望竞争比下界，这一结果由 Yao 原理[6]得出。

当至少有两台机器时，完全在线问题比半在线问题更为一般。Albers 和 Eckl[4]提出了在列表调度规则[7]中测试作业 J_j，即当 $r_j\ge\varphi$ 时，按照该规则将每个作业分配到最空闲的机器上进行处理（可能是测试后执行）。他们证明了这样的算法是 $\varphi(2-\frac{1}{m})$-竞争的，且该分析是紧的，其中 m 是机器的数量。他们还证明了，即使在均匀测试情况下 $P|\text{online},t_j=1,0\le p_j\le u_j|C_{\max}$ 中，任何确定性算法的竞争比下界为 2[4]，并且对于两台机器的一般测试情况 $P_2|\text{online},t_j,0\le p_j\le u_j|C_{\max}$，即当只有两台机器时，其下界为 2.0953，这一结果也稍微更好[4]。

1.2 我们的成果

对于完全在线和半在线的多处理器调度与测试问题，目标是最小化最大完工时间，我们已经看到，现有的随机化算法仅为单机器情况下的最优期望 $\frac{4}{3}$ 竞争算法[1-3]。它们的期望竞争比严格小于任何确定性算法竞争比下界 φ。对于经典的在线多处理器调度问题，只有当 $2\le m\le 5$ 时，Bartal 等人[8] 和 Seiden[9] 提出的算法的期望竞争比严格小于任何确定性算法的竞争比下界；Albers[10] 提出的算法期望竞争比 1.916 也严格小于确定性下界 1.919，但条件是只使用了三种离线最大完工时间下界。

在本文中，我们始终假设完全在线问题 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 中至少有两台机器。我们提出了一个几乎随机化的算法，它是任意多个组件确定性算法的集成，每个算法以一定的概率运行，并证明其期望竞争比最多为 $\sqrt{\left(1-\frac{1}{m}\right)^2\varphi^2+2\left(1-\frac{1}{m}\right)+1}$, 其中 m 是机器的数量。这个期望竞争比随着 m 的增大而严格递增，严格小于 Albers 和 Eckl[4]提出的最佳已知确定性算法的竞争比 $\varphi(2-\frac{1}{m})$，并且当 m 趋于无穷大时，它逼近 $(\sqrt{\varphi+3}+1)\approx 3.1490$。据我们所知，文献中的几乎随机化算法大多数是其组件确定性算法的均匀分布，而我们的算法则是第一个非均匀分布的任意多个组件算法。

当只有两台机器时，即对于 $P_2|\text{online},t_j,0\le p_j\le u_j|C_{\max}$，我们展示了在随机化算法中采用两个组件确定性算法导致的期望竞争比为 $\frac{3\varphi+3\sqrt{13-7\varphi}}{4}\approx 2.1839$

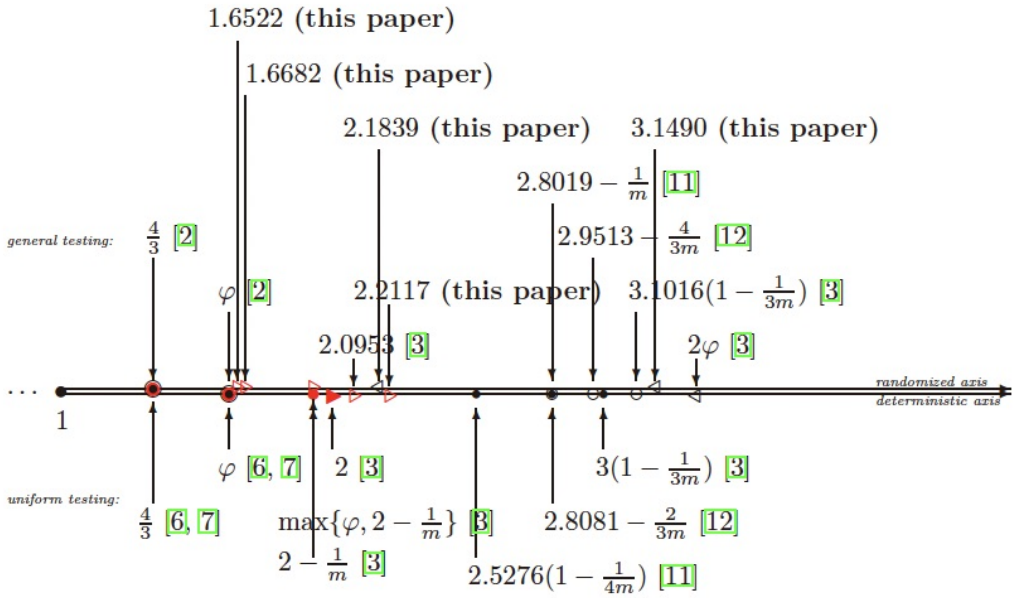

图 3.1 现有的确定性和随机化算法在完全在线和半在线多处理器调度与测试问题中的竞争比。在横轴上，红色符号表示下界；三角形（◁和▷表示一般测试，否则表示均匀测试）表示完全在线问题的结果，而圆形（○表示一般测试，·表示均匀测试）表示半在线问题的结果。在两个横轴之间，底部的轴表示确定性算法，顶部的轴表示随机化算法；轴上方的结果是针对一般测试情况的，而轴下方的结果是针对均匀测试情况的。

.

对于这个两台机器的情况，Albers 和 Eckl[4] 证明了任何确定性算法的竞争比下界为 2.0953。我们将其改进为 2.2117，从而证明我们的随机化算法在期望竞争比方面优于任何确定性算法。

关于不可近似性，我们证明了至少三台机器时，任何随机化算法的期望竞争比下界为 1.6682，而在仅有两台机器时，对于 $P_2|\text{online},t_j,0\le p_j\le u_j|C_{\max}$，期望竞争比的下界为 1.6522，这两个下界都是使用稍微修改过的 Yao 原理[6] 得出的。这两个下界在三台和两台机器情况下严格优于 Albers 和 Eckl[4] 提出的下界 $2-\frac{1}{m}$。我们的所有结果总结如图 3.1所示。

本文其余部分的组织结构如下：第 2 节介绍一些基本的符号和定义；接下来，在第 3 节中，我们证明了至少三台机器时完全在线问题 $P|t_j,0\le p_j\le u_j|C_{\max}$ 的期望竞争比下界 1.6682 和仅两台机器时的下界 1.6522；第 4 节包含了该问题的随机化算法及其性能分析、两台机器情况下稍微修改的随机化算法及其性能分析，以及对于 $P_2|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 的任何确定性算法的竞争比下界 2.2117；最后，第 5 节对论文进行了总结。

2 预备内容

我们研究完全在线的多处理器调度与测试问题，以最小化最大完工时间，记作 $P|\text{online},t_j,0\leq p_j\leq u_j|C_{\max}$，其中作业依次到达。我们的目标是设计期望竞争比优于现有最先进的确定性算法的随机化算法，或者更进一步，优于已证明的最佳确定性竞争比下界。所有作业都在时间零到达的特殊情况称为半在线问题，记作 $P|t_j,0\leq p_j\leq u_j|C_{\max}$。在适用的情况下，我们将证明半在线变体的下界。

多处理器调度与测试的一个实例 I 包含一组作业集 $J=\{J_1,J_2,\ldots,J_n\}$，每个作业都要在一组 m 台并行的相同机器 $M=\{M_1,M_2,\ldots,M_m\}$ 上执行，其中 n 和 $m\geq 2$ 是输入的一部分。每个作业 J_j 都有一个已知的处理时间上界 u_j 和一个长度为 t_j 的测试操作，处理时间 p_j 在测试操作执行之前是未知的。也就是说，调度器可以选择不测试作业 J_j，而是执行它 u_j 时间，或者选择测试它 t_j 时间后，在同一台机器上立即执行 p_j 时间。

确定性算法 A 对是否测试作业做出二元决策。令 p_{A_j} 和 ρ_j 分别表示作业 J_j 在算法 A 和最优离线调度中的总时间。可以看到，当作业被测试时，$p_{A_j}=t_j+p_j$，否则 $p_{A_j}=u_j$，而 $\rho_j=\min\{u_j,t_j+p_j\}$。令 C_j 表示作业 J_j 在算法 A 生成的调度中的完成时间。问题的目标是最小化最大完工时间 $C_{\max}=\max_j C_j$。我们使用 $C_A(I)$（当实例 I 在上下文中明确时，简称为 C_A）和 $C^*(I)$（或相应的 C^*）分别表示算法 A 和最优离线调度在实例 I 上的最大完工时间。

切换到更一般的随机化算法 A，其在实例 I 上的期望最大完工时间表示为 $E[C_A(I)]$。

定义 2.1. 竞争比：对于确定性算法 A，竞争比是算法 A 生成的调度的最大完工时间与最优离线调度的最大完工时间之间的最坏情况比值，即 $\sup_I\left\{\frac{C_A(I)}{C^*(I)}\right\}$ 其中 I 遍历所有问题实例。

对于更一般的随机化算法A，其期望竞争比为$\sup_I\left\{\frac{E[C_A(I)]}{C^*(I)}\right\}$

由于作业 J_j 的处理时间 p_j 可能达到 0 和 u_j 两个极端值，确定性算法通常会设置一个阈值函数 $r_j=\frac{u_j}{t_j}$ 用于二元测试决策。直观上，随机化算法应该以概率 $f(r_j)$ 测试作业 J_j，并且该概率函数应该随着比率 r_j 的增大而增大，且当 $r_j\leq 1$ 时，$f(r_j)=0$。

实际上，D¨urr 等人[1-2] 和 Albers 与 Eckl[3] 在他们的最优期望 4/3 竞争随机化算法中使用了如下的概率函数：

$$f(r)=\begin{cases}0, & \text{如果} r\le 1,\\ \frac{r(r-1)}{r(r-1)+1}, & \text{如果} r>1.\end{cases} \tag{3-1}$$

他们分别应用于单机器问题 $P_1|\text{online},t_j=1,0\le p_j\le u_j|C_{\max}$ 和 $P_1|\text{online},t_j,0\le p_j\le u_j|C_{\max}$。遗憾的是，他们的成功不能轻易扩展到多机器情况，因为他们性能分析中使用的一个关键事实是，对于单台机器，期望最大完工时间等于所有作业期望处理时间的总和。我们在下面展示，对于多台机器，如果一个随机化算法使用式 3-1 中的概率函数来测试作业，那么当机器数量趋近于无穷大时，它的期望竞争比将是无界的。

引理 2.2. 对于 $P|online,t_j=1,0\le p_j\le u_j|C_{max}$，如果一个随机化算法使用式 *3-1* 中的概率函数来测试作业，那么当机器数量趋近于无穷大时，它的期望竞争比将是无界的。

引理 2.2 的证明在某种程度上表明，当作业 J_j 的比率 r_j 足够大时，应该测试作业，而不是仅仅为不测试留下一点概率。事实上，稍后我们将看到，在我们提出的随机化算法中，对于绝对测试作业，有一个关于 r_j 的阈值。另一方面，在至少有两台机器的情况下，由于随机作业测试决策的存在，我们不再能确定每个作业分配给哪台机器。如果使用如式 3-1 中的作业测试概率函数，调度决策变得随机。因此，我们摒弃了[1-3] 中的随机化算法设计思想，而是扩展了 Albers[10] 的思想，设计了多个相互补充的组件确定性算法 $A_0,A_1,\ldots,A_\ell$，其中 ℓ 可以任意大，然后以一定的概率 α_i 运行每个 A_i。令 $p^j_{A_i}$ 表示作业 J_j 在算法 A_i 中消耗的总时间，对于任意 $i=0,1,\ldots,\ell$。随机化算法 A 中作业 J_j 的期望处理时间为

$$E[p^j_A]=\sum_{i=0}^{\ell}\alpha_i p^j_{A_i}. \tag{3-2}$$

类似地，令 C_{A_i} 为算法 A_i 生成的调度中的最大完工时间，对于任意 $i=0,1,\ldots,\ell$。期望最大完工时间为

$$E[C_A]=\sum_{i=0}^{\ell}\alpha_i C_{A_i}. \tag{3-3}$$

以下四个引理对于任何以概率分布 $(\alpha_0,\alpha_1,\ldots,\alpha_\ell)$ 运行的组件确定性算法 $A_0,A_1,\ldots,A_\ell$ 均成立，同时假设在不失一般性的情况下，它们都不会对任何作业 J_j 在 $r_j\le 1$ 时进行测试。

引理 2.3. 对于任意 $i=0,1,\ldots,\ell$，如果作业 J_j 在 A_i 中被测试且 $r_j>1$，则

$$p_{A_i}^j \le \left(1+\frac{1}{r_j}\right)\rho_j.$$

引理 2.4. 对于任意 $i=0,1,\ldots,\ell$，如果作业 J_j 在 A_i 中未被测试且 $r_j>1$，则

$$p_{A_i}^j \le r_j\rho_j.$$

引理 2.5. 如果作业 J_j 在一组算法 T 中被测试，但在任何 $\{A_0,A_1,\ldots,A_\ell\}\setminus T$ 中未被测试，则

$$E[p_A^j] \le \max\left(\theta+(1-\theta)r_j, 1+\frac{\theta}{r_j}\right)\rho_j,$$

其中 $\theta=\sum_{A_i\in T}\alpha_i$。

引理 2.6. 以下两个最优离线最大完工时间 C^* 的下界成立：

$$C^* \ge \max\left\{\frac{1}{m}\sum_{j=1}^{n}\rho_j,\quad \max_{j=1}^{n}\rho_j\right\}. \tag{3-4}$$

在完全在线问题 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 中，作业依次到达，假设作业的顺序为 $hJ_1,J_2,\ldots,J_n$。在半在线问题 $P|t_j,0\le p_j\le u_j|C_{\max}$ 中，所有作业都在时间零到达，算法可以利用所有已知的 u_j 和 t_j 来按任意顺序处理它们。

3 期望竞争下界

定理 3.1. 对于任何随机算法 A_r 和任何随机实例 I_r，A_r 的期望竞争比满足

$$\sup_{I\in I}\frac{E[C_{A_r}(I)]}{C^*(I)} \ge \inf_{A\in A}\frac{E[C_A(I_r)]}{E[C^*(I_r)]},$$

其中 $E[C_A(I_r)]$ 和 $E[C^*(I_r)]$ 分别是确定性算法 A 在实例 I_r 上的期望目标值和最优期望目标值。

对于半在线单机均匀测试问题 $P1|t_j=1,0\le p_j\le u_j|C_{\max}$，以下随机实例展示了期望竞争比的紧致下界 $\frac{4}{3}$[1][2][3]。

例子 3.2. 在这个 $P1|t_j=1,0\le p_j\le u_j|C_{\max}$ 的随机实例中，只有一个任务 J_1，其参数为 $u_1=2$ 和 $t_1=1$，且 $p_1=0$ 或 2 的概率各为 *0.5*。

在任何确定性算法 A 中，J_1 要么被测试，要么未被测试。如果 J_1 未被测试，则 $p_1^A = u_1 = 2$；否则 J_1 被测试，因此 $p_1^A = t_1 + p_1$，这可能是 *1* 或 *3*，每种情况的概率各为 *0.5*。由此得出期望总执行时间为 $E[p_1^A] = 2$，无论 J_1 是否被测试。

另一方面，最优离线执行时间是 $\rho_1 = 1$ 如果 $p_1 = 0$，或者 $\rho_1 = 2$ 如果 $p_1 = 2$；即，期望最优执行时间为 $E[\rho_1] = 1.5$。根据定理 *3.1*，$\frac{E[p_1^A]}{E[\rho_1]} = \frac{4}{3}$ 是半在线单机均匀测试问题 $P1|t_j = 1, 0 \le p_j \le u_j|C_{\max}$ 的期望竞争比的下界。

当有多台机器时，以下确定性实例展示了期望竞争比的下界 $2 - \frac{1}{m}$[4]。

例子 3.3. 在这个 $P|t_j = 1, 0 \le p_j \le u_j|C_{\max}$ 的确定性实例中，有 m 台机器和 m 个任务 $J_1, J_2, \ldots, J_m$，每个任务的参数为 $u_j = 2$ 和 $t_j = 1$，且 $p_j = 0$ 或 2 的概率各为 *0.5*。

在任何确定性算法 A 中，每个任务 J_j 要么被测试，要么未被测试。如果 J_j 未被测试，则 $p_j^A = u_j = 2$；否则 J_j 被测试，因此 $p_j^A = t_j + p_j$，这可能是 *1* 或 *3*，每种情况的概率各为 *0.5*。由此得出每个任务的期望执行时间为 $E[p_j^A] = 2$。

考虑最优离线调度。如果所有任务 J_j 都未被测试，则每个任务的实际处理时间为 *2*，总完成时间为 $2m$。如果所有任务都被测试，则每个任务的实际处理时间是 *1* 或 *3*，每种情况的概率各为 *0.5*，总完成时间为 $m \times 1.5 = 1.5m$。最优离线调度可以通过将任务分配到不同的机器上来实现最小的完成时间。最优离线完成时间为 $\rho = \frac{2m}{m} = 2$ 如果所有任务都未被测试，或者 $\rho = \frac{1.5m}{m} = 1.5$ 如果所有任务都被测试。因此，期望最优完成时间为 $E[\rho] = 1.5$。

根据定理 *3.1*，对于多台机器的情况，$\frac{E[C_A]}{E[\rho]} = \frac{2m}{1.5m} = \frac{4}{3}$ 是一个下界。然而，通过更精细的分析，可以得到更紧致的下界 $2 - \frac{1}{m}$。

因此，对于多台机器的情况，期望竞争比的下界为 $2 - \frac{1}{m}$。

例子 3.4. 在这个 $P|online, t_j, 0 \le p_j \le u_j|C_{\max}$ 的确定性实例 I 中，有 m 台机器和 $m+1$ 个任务按顺序到达 $\langle J_1, J_2, \ldots, J_{m+1} \rangle$，每个任务的参数为

$$u_j = 2, \quad t_j = 1, \quad \text{对于任何} j = 1, 2, \ldots, m+1.$$

也就是说，这 $m+1$ 个任务在到达时是不可区分的。对于一个确定性算法，对手将前两个未被测试的任务（如果存在）或序列中的最后两个任务的实际执行时间设置为 *0*；其他 $m-1$ 个任务的实际执行时间为 *2*。

例如，当 $m=4$ 且算法不测试任何任务时，实际执行时间序列为 $\langle 0,0,2,2,2\rangle$；如果算法不测试 J_3,J_4 和 J_5 中的任何一个，则实际执行时间序列为 $\langle 2,2,0,0,2\rangle$；如果算法仅不测试 J_1，则实际执行时间序列为 $\langle 0,2,2,2,0\rangle$；如果算法测试所有五个任务，则实际执行时间序列为 $\langle 2,2,2,0,0\rangle$。

给定一个确定性算法 A，记 $C^A_{\min}(I)$ 为算法 A 在实例 I 上产生的调度中的最小机器负载。

引理 3.5. 当 $m\geq 3$ 时，在实例 *3.4* 中的实例 I 上，对于任何确定性算法 A，要么 $C_A(I)\geq 4$，要么 $C_A(I)=3$ 且 $C_{\min}(I)\geq 2$。

证明. 如果算法 A 不测试两个或更多任务，则每个任务的总处理时间至少为 2。根据鸽巢原理，这会导致完成时间 $C_A(I)\geq 4$。如果算法 A 测试了所有任务中的至多一个任务，则在前 m 个任务 $\langle J_1,J_2,\ldots,J_m\rangle$ 中，除了一个例外任务外，每个任务的总处理时间为 3。这个例外任务的总处理时间如果是被测试的则为 1，未被测试的则为 2。可以看到，当一台机器被分配了这些 m 个任务中的任意两个时，根据鸽巢原理，完成时间 $C_A(I)\geq 4$。在另一种情况下，每台机器恰好被分配了一个这些 m 个任务中的任务，将最后一个任务 J_{m+1} 分配给任何一台机器会导致要么 $C_A(I)\geq 4$，要么 $C_A(I)=3$ 且 $C_{\min}(I)\geq 2$。 □

引理 3.6. 当 $m=2$ 时，在实例 *3.4* 中的实例 I 上，对于任何确定性算法 A，要么 $C_A(I)\geq 5$，要么 $C_A(I)\geq 4$ 且 $C_{\min}(I)\geq 1$，要么 $C_A(I)\geq 3$ 且 $C_{\min}(I)\geq 2$。

证明. 设 I 是一个三任务实例。我们区分两种情况：算法 A 是否测试第一个任务 J_1。

1. J_1 被测试。在这种情况下，$p_1=2$，从而 $p^A_1=3$。注意到 J_2 和 J_3 的总处理时间至少为 1。这三个值 $\{3,1,1\}$ 保证了其中一个完成时间场景。

2. J_1 未被测试。在这种情况下，$p_1=0$，从而 $p^A_1=2$。注意到 J_2 和 J_3 的总处理时间至少为 2。这三个值 $\{2,2,2\}$ 也保证了其中一个完成时间场景。

因此，无论哪种情况，都可以保证完成时间 $C_A(I)\geq 4$ 或 $C_A(I)=3$ 且 $C_{\min}(I)\geq 2$。

如果 J_1 未被测试，则 $p^A_1=2$。注意到 J_2 和 J_3 中的一个任务的总处理时间至少为 2（要么未被测试，要么被测试且实际执行时间为 2），另一个任务的总处理时间至少为 1。这三个值 $\{2,2,1\}$ 保证了其中一个完成时间场景。 □

定理 3.7. 对于存在三台或更多机器的情况 $P|online,t_j,0\le p_j\le u_j|C_{\max}$，任何随机算法的期望竞争比至少为 $\frac{21}{2}-\sqrt{78}\approx 1.6682$。

定理 3.8. 对于问题 $P2|online,t_j,0\le p_j\le u_j|C_{\max}$，任何随机算法的期望竞争比至少为 $\frac{21+4\sqrt{51}}{30}\approx 1.6522$。

4 随机算法

在本节中，我们首先提出了一种针对至少两台机器存在的完全在线多处理器调度与测试问题 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$ 的随机算法，并证明其期望竞争比为 $(\sqrt{\varphi}+3+1)\approx 3.1490$。当只有两台机器时，我们稍微调整了算法中的参数，从而得到了改进的期望竞争比 $\frac{3\varphi+3\sqrt{13-7\varphi}}{4}\approx 2.1839$。最后，我们证明了对于两台机器的情况，任何确定性算法的竞争比的改进下界为 2.2117，这意味着我们的随机算法在期望竞争比方面优于任何确定性算法。

对于完全在线多处理器调度与测试问题 $P|\text{online},t_j,0\le p_j\le u_j|C_{\max}$，Albers 和 Eckl[3] 提出如果 $r_j\ge\varphi$ 则对任务 J_j 进行测试，在列表调度规则[7] 中将每个任务分配给负载最小的机器进行处理（可能先测试后执行）。他们证明了这种确定性算法是紧致的 $\varphi(2-\frac{1}{m})$-竞争比，其中 m 是机器的数量。我们将它作为我们的第一个组合算法 A_0，并以概率 $\alpha_0=\alpha(m,\ell)$ 运行该算法，如公式 3-5 所示，其中 $\ell\ge 1$ 是由我们的随机算法选择的一个固定常数。

$$\alpha_0=\alpha(m,l)=\sqrt{\frac{(1-\frac{1}{m})(l+1)\varphi^2}{(1-\frac{1}{m})(l+1)\varphi^2+2l}},a_i=\beta(m,l)=\frac{1-\alpha(m,l)}{l},i=1,2,\ldots,l. \tag{3-5}$$

以下 $\alpha(m,\ell)$ 和 $\beta(m,\ell)$ 分别简写为 α 和 β。

可以看到，当 $r_j\le 1$ 时，任务 J_j 不应该被测试，这在算法 A_0 中也是如此。另一方面，当 r_j 很大时，即使 p_j 非常接近 u_j，测试也不会浪费太多时间，这也与算法 A_0 的情况一致。“做出错误测试决策的风险”发生在 r_j 接近 φ 时，因为在这种情况下，如果 J_j 未被测试，则 p_j 可能非常接近 0；而如果被测试，则 p_j 可能非常接近 u_j。然而，从引理 2.2 的证明中我们也可以看到，即使对具有较大比率 r_j 的任务进行不测试的概率很小，也可能导致期望竞争比无界，特别是在机器数量 m 趋于无穷大时。

因此，在其他每个组合算法 A_i（$i=1,2,\ldots,\ell$）中，我们设置一个阈值 $y_i(m,\ell)>\varphi$ 如公式 3-6 所示，使得比率 $r_j\ge y_i(m,\ell)$ 的任务 J_j 在 A_i 中被测试；然后我们设置另一个阈值

$x_i(m,\ell)=1+\frac{1}{y_i(m,\ell)}$，使得比率 $r_j \le x_i(m,\ell)$ 的任务 J_j 在 A_i 中不被测试。

$$y_i(m,l)=\frac{\varphi(\alpha+i\beta)}{\alpha}, x_i(m,l)=1+\frac{1}{y_i(m,l)}, i=0,1,\ldots,l. \tag{3-6}$$

类似地，下面我们将 $y_i(m,\ell)$ 和 $x_i(m,\ell)$ 简化为 y_i 和 x_i，分别表示。当 $r_j \in (x_i,y_i)$ 时，我们在 A_i 中反转测试决策，即如果 $r_j \le \varphi$ 则测试 J_j，如果 $r_j > \varphi$ 则不测试 J_j。我们的随机算法，记作 GCL，以概率 α_i 选择运行 A_i，其中 $i=0,1,\ldots,\ell$。

从公式 3-5 和 3-6 可以看出，$\alpha,\beta>0$，并且

$$1<x_\ell<\ldots<x_1<x_0=\varphi=y_0<y_1<\ldots<y_\ell=\frac{\varphi}{\alpha}. \tag{3-7}$$

算法 GCL 的简要描述如图 2 所示。

输入: m 台机器，以及按顺序到达的 n 个任务 $\langle J_1,J_2,\ldots,J_n\rangle$;

输出: 一个调度方案，其中每个任务被测试或不被测试，并指定处理该任务的机器。

步骤:

1. 选择 ℓ，并根据公式 3-5 和 3-6 设置参数 α_i,y_i 和 x_i，对于 $i=0,1,\ldots,\ell$;
2. 组成算法 A_i，$i=0,1,\ldots,\ell$: 对于每个任务 J_j，
   (a) 如果 $r_j \le x_i$ 或 $\varphi < r_j \le y_i$，则将 J_j 未测试地分配到负载最小的机器上进行处理;
   (b) 否则，将 J_j 分配到负载最小的机器上进行测试和处理。
3. 以概率 α_i 选择运行 A_i，对于 $i=0,1,\ldots,\ell$。

引理 4.1.

$$\frac{1-\alpha}{x_i}<\frac{\alpha}{\varphi}, (l-i)\beta(y_{i+1}-1)<\frac{\alpha}{\varphi}, \forall i=0,1,\ldots,\ell-1.$$

引理 4.2. 在算法 A_i 中，$p_j^{A_i} \le y_i\rho_j, \forall J_j, i=0,1,\ldots,\ell$。

引理 4.3. 在算法 *GCL* 中，$E[p_j^A] \le x_l\rho_j, \forall J_j$。

定理 4.4. 对于问题 $P|online,t_j,0\le p_j\le u_j|C_{\max}$，若设置 $\alpha(m,\ell)$ 和 $\beta(m,\ell)$ 如公式 *3-5* 所示，并且对于每一个 $i=0,1,\ldots,\ell$ 设置 $y_i(m,\ell)$ 和 $x_i(m,\ell)$ 如公式 *3-6* 所示，其中 $m\ge 2$ 是机器

的数量，$\ell \geq 1$ 是一个固定的常数，则算法 *GCL* 的期望竞争比最多为

$$\sqrt{(1-\frac{1}{m})^2(1+\frac{1}{\ell})^2\varphi^2+2(1-\frac{1}{m})(1+\frac{1}{\ell})}+1-(1-\frac{1}{m})\frac{\varphi}{\ell}$$

通过选择一个足够大的 ℓ，算法 GCL 的期望竞争比接近 $\sqrt{(1-\frac{1}{m})^2\varphi^2+2(1-\frac{1}{m})}+1$ 该值随着 m 的增加而增加，并且当 m 趋于无穷大时，接近 $\sqrt{\varphi+3}+1\approx 3.1490$。相应地，可以验证 α, y_ℓ 和 x_ℓ 分别接近 $\frac{\sqrt{\varphi}+1}{\sqrt{\varphi+3}}\approx 0.7529,\quad \sqrt{\varphi}+3\approx 2.1490,\quad 1+\frac{1}{\sqrt{\varphi+3}}\approx 1.4653.$

在算法 GCL 中，我们可以将任务的测试概率 $f(r)$ 作为其比率 r 的函数来确定。例如，如果 $r \geq y_\ell$，则 $f(r)=1$；如果 $r \leq x_\ell$，则 $f(r)=0$。对于一个固定的常数 ℓ，可以看到这种概率函数 $f(r)$ 是阶梯状且递增的。当 ℓ 趋于无穷大时，这种概率函数的极限（仍然记作 $f(r)$）是有趣的。首先，注意到 $\alpha(m,\ell)$ 接近 $\alpha(m)=\sqrt{\frac{(1-\frac{1}{m})\varphi^2}{(1-\frac{1}{m})\varphi^2+2}}$，$y_\ell(m,\ell)$ 和 $x_\ell(m,\ell)$ 分别接近 $y(m)=\frac{\varphi}{\alpha(m)}$ 和 $x(m)=1+\frac{1}{y(m)}$。然后，根据公式 3-6，我们有

$$f(r)=\begin{cases}0, & \text{当} r \leq x(m),\\ 1-\frac{\alpha(m)}{\varphi(r-1)}, & \text{当} x(m)<r\leq\varphi,\\ \frac{\alpha(m)r}{\varphi}, & \text{当} \varphi<r\leq y(m),\\ 1, & \text{当} r>y(m),\end{cases}$$

该函数是递增的，并且除了在 φ 处不连续外，在其他地方都是连续的。

当只有两台机器时，即 $m=2$，通过选择一个足够大的 ℓ，算法 GCL 对于问题 $P2|\text{online},t_j,0\leq p_j\leq u_j|C_{\max}$ 已经是期望 $\frac{1}{2}\sqrt{\varphi+5}+1\approx 2.2863$-竞争比。注意，在算法内部，选择算法 A_0 的概率接近 $\alpha(2)=\frac{\varphi}{\sqrt{\varphi+5}}\approx 0.6290$。

接下来我们展示，通过选择另一个运行 A_0 的概率，基于两个组成算法的修订后的 GCL 算法对于问题 $P2|\text{online},t_j,0\leq p_j\leq u_j|C_{\max}$ 是期望 $\frac{3\varphi+3\sqrt{13-7\varphi}}{4}\approx 2.1839$-竞争比。进一步地，这个期望竞争比的界是紧的。

我们注意到，对于问题 $P|\text{online},t_j,0\leq p_j\leq u_j|C_{\max}$，我们的 GCL 算法的期望竞争比优于已知的最佳确定性竞争比 $\varphi(2-\frac{1}{m})$，但远高于确定性竞争比的下界 2[4]。它更远离定理 3.8 中的 1.6522、定理 3.7 中的 1.6682 以及在两台、三台和 $m\geq 4$ 台机器存在的情况下期望竞争比的下界 $2-\frac{1}{m}$[4]。

回顾一下，除了对均匀测试情况 $P|\text{online},t_j=1,0\leq p_j\leq u_j|C_{\max}$ 的确定性竞争比下界 2 之外，Albers 和 Eckl[4] 还展示了两台机器一般测试情况 $P2|\text{online},t_j,0\leq p_j\leq u_j|C_{\max}$ 的稍

好一些的下界 2.0953。我们在下一个定理中将该下界从 2.0953 改进到 2.2117。因此，对于 $P2|\text{online},t_j,0\le p_j\le u_j|C_{\max}$，GCL 的期望竞争比不仅优于半在线变体 $P2|t_j,0\le p_j\le u_j|C_{\max}$ 的已知最佳确定性竞争比 2.3019，而且优于完全在线问题的确定性竞争比下界 2.2117。

定理 4.5. 对于问题 $P2|online,t_j,0\le p_j\le u_j|C_{\max}$，任何确定性算法的竞争比大于 *2.2117*。

证明. 使用对抗性论证来构造一个三任务实例，任务顺序为 $\langle J_1,J_2,J3\rangle$。考虑一个确定性算法 A。

第一个任务 J_1 的参数为 $u_1=\varphi$ 和 $t_1=1$。如果 A 测试 J_1，则对手设置 $p_1=\varphi$。因此 $p_1^A=\varphi+1$ 且 $\rho_1=\varphi$。如果 J_1 未测试，则对手设置 $p_1=0$，从而 $p_1^A=\varphi$ 且 $\rho_1=1$。无论哪种情况，都有 $\frac{p_1^A}{\rho_1}=\varphi$，因此我们假设 $p_1^A=\varphi$ 且 $\rho_1=1$。（如果 $p_1^A=\varphi+1$ 且 $\rho_1=\varphi$，则下面与 J_2 和 J_3 相关的 u_j 和 t_j 值将乘以一个因子 φ 。）

对于任何即将到来的任务 J_j（即 $j=2,3$，对手总是设置 $p_j=u_j$ 如果 J_j 被算法测试，否则设置 $p_j=0$。

设 $x_0=\frac{3\varphi+1-\sqrt{11\varphi+6}}{2}\approx 0.4878$，这是二次方程 $x^2-(3\varphi+1)x+\varphi^2=0$ 的一个根。第二个任务 J_2 的参数为 $u_2=\varphi-x_0$ 和 $t_2=x_0$。我们区分两种情况：J_2 是否被算法测试。

情况 1: J_2 被算法测试。在这种情况下，$p_2=u_2$ 且 $p_2^A=\varphi$ 且 $\rho_2=\varphi-x_0$。如果 J_2 被安排在与 J_1 同一台机器上，则第三个任务 J_3 被取消（通过设置 $u_j=t_j=0$），导致 $C_A=p_1^A+p_2^A=2\varphi$。注意到 $\rho_2>1$ 且 $C^*=\rho_2=\varphi-x_0$。因此竞争比为 $\frac{2\varphi}{\varphi-x_0}$。

下面假设 J_j 被分配到机器 M_j 上，对于 $j=1,2$，每台机器的负载为 φ。第三个任务 J_3 的参数为 $u_3=y_0$ 和 $t_3=\varphi+1-x_0$，其中 $y_0=\frac{1-x_0+\sqrt{(3\varphi-5)x_0+15\varphi+8}}{2}\approx 3.0933$ 是二次方程 $y^2-(1-x_0)y-(\varphi+1-x_0)(2\varphi+1-x_0)=0$ 的一个根。

如果 J_3 未被测试，则 $p_3=0$，从而 $p_3^A=y_0$ 且 $\rho_3=\varphi+1-x_0$，导致 $C_A=\varphi+y_0$。在最优离线调度中，J_1 和 J_2 被安排在一台机器上，而 J_3 被安排在另一台机器上，导致最优离线完成时间为 $C^*=\varphi+1-x_0$。因此，竞争比为 $\frac{C_A}{C^*}=\frac{y_0+\varphi}{\varphi+1-x_0}$。

如果 J_3 被测试，则 $p_3=u_3$，从而 $p_3^A=t_3+u_3=\varphi+1-x_0+y_0$ 且 $\rho_3=y_0$，导致 $C_A=2\varphi+1-x_0+y_0$。在最优离线调度中，J_1 和 J_2 被安排在一台机器上，而 J_3 被安排在另一台机器上，导致 $C^*=y_0$，因此，竞争比为 $\frac{C_A}{C^*}=\frac{2\varphi+1-x_0+y_0}{y_0}$。

综上所述，在这种情况下，我们有

$$\frac{C_A}{C^*} \geq \min\left\{\frac{2\varphi}{\varphi - x_0}, \frac{y_0+\varphi}{\varphi+1-x_0}, \frac{2\varphi+1-x_0+y_0}{y_0}\right\}.$$

由于 y_0 是二次方程 $y^2-(1-x_0)y-(\varphi+1-x_0)(2\varphi+1-x_0)=0$ 的一个根，我们有

$$y_0(y_0+\varphi) = (\varphi+1-x_0)(y_0+2\varphi+1-x_0),$$

即上述最后两个量相等且大约为 2.21172。第一个量约为 2.8634。因此，$\frac{C_A}{C^*} > 2.2117$。

情况 2: J_2 未被算法测试。在这种情况下，$p_2=0$，从而 $p_2^A=\varphi-x_0$ 且 $\rho_2=x_0$。如果 J_2 被安排在与 J_1 同一台机器上，则第三个任务 J_3 被取消（通过设置 $u_j=t_j=0$），导致 $C_A=p_1^A+p_2^A=2\varphi-x_0$。注意到 $\rho_2=x_0<1$ 且 $C^*=1$。因此，竞争比为 $\frac{C_A}{C^*}=2\varphi-x_0$。

下面假设 J_j 被分配到机器 M_j 上，对于 $j=1,2$，两台机器的负载分别为 φ 和 $\varphi-x_0$。第三个任务 J_3 的参数为 $u_3=\frac{(1+\varphi)y_0}{2\varphi+1-x_0}\approx 2.1606$ 和 $t_3=1+x_0$，其中 y_0 与情况 1 相同。

如果 J_3 未被测试，则 $p_3=0$，从而 $p_3^A=u_3=\frac{(1+\varphi)y_0}{2\varphi+1-x_0}$ 且 $\rho_3=1+x_0$，无论 J_3 被分配到哪台机器上，都有 $C_A\geq\varphi-x_0+\frac{(1+\varphi)y_0}{2\varphi+1-x_0}$。在最优离线调度中，$J_1$ 和 J_2 被安排在一台机器上，而 J_3 被安排在另一台机器上，导致最优离线完成时间为 $C^*=1+x_0$。因此，竞争比为

$$\frac{C_A}{C^*} \geq \frac{\varphi-x_0+\frac{(1+\varphi)y_0}{2\varphi+1-x_0}}{1+x_0} = \frac{(\varphi-x_0)(2\varphi+1-x_0)+(1+\varphi)y_0}{(1+x_0)(2\varphi+1-x_0)}.$$

由于 x_0 是二次方程 $x^2-(3\varphi+1)x+\varphi^2=0$ 的一个根，我们有

$$(\varphi-x_0)(2\varphi+1-x_0) = x_0^2-(3\varphi+1)x_0+\varphi(1+2\varphi) = \varphi(1+\varphi)$$

和

$$(1+x_0)(2\varphi+1-x_0) = -x_0^2+2\varphi x_0+2\varphi+1 = (\varphi+1)(1+\varphi-x_0).$$

因此，$\frac{C_A}{C^*}\geq\frac{y_0+\varphi}{1+\varphi-x_0}$。

如果 J_3 被测试，则 $p_3=u_3$，从而 $p_3^A=t_3+u_3=1+x_0+\frac{(1+\varphi)y_0}{2\varphi+1-x_0}$ 且 $\rho_3=\frac{(1+\varphi)y_0}{2\varphi+1-x_0}$，无论 J_3 被分配到哪台机器上，都有 $C_A\geq 1+\varphi+\frac{(1+\varphi)y_0}{2\varphi+1-x_0}$。由于 $\rho_3>\rho_1+\rho_2$，在最优离线调度中，J_1 和 J_2 被安排在一台机器上，而 J_3 被安排在另一台机器上，导致 $C^*=\rho_3=\frac{(1+\varphi)y_0}{2\varphi+1-x_0}$，进而竞争比为

$$\frac{C_A}{C^*} \geq \frac{1+\varphi+\frac{(1+\varphi)y_0}{2\varphi+1-x_0}}{\frac{(1+\varphi)y_0}{2\varphi+1-x_0}} = \frac{y_0+2\varphi+1-x_0}{y_0}.$$

综上所述，在这种情况下，我们有

$$\frac{C_A}{C^*} \geq \min\left\{2\varphi - x_0, \frac{y_0+\varphi}{1+\varphi-x_0}, \frac{2\varphi+1-x_0+y_0}{y_0}\right\}.$$

与情况 1 相同，上述最后两个量相等且大约为 2.21172。第一个量约为 2.7482。因此，$\frac{C_A}{C^*} > 2.2117$。

以上两种情况证明了对于问题 $P2|\text{online},t_j,0 \leq p_j \leq u_j|C_{\max}$，任何确定性算法的竞争比大于 2.2117。

□

5 结论

我们研究了完全在线多处理器调度与测试问题 $P|\text{online},t_j,0 \leq p_j \leq u_j|C_{\max}$，并提出了一种随机算法 GCL，该算法是非均匀分布的任意多个确定性算法。据我们所知，文献中的随机算法作为元算法大多是其组成确定性算法的均匀分布，而我们的 GCL 偏向于其中一个组成算法；此外，当有多个组成算法时，以前的随机算法通常需要记录所有解，而我们的 GCL 不需要这样做，因为组成算法彼此独立。我们证明了 GCL 的期望竞争比约为 3.1490。当只有两台机器时，即对于 $P2|\text{online},t_j,0 \leq p_j \leq u_j|C_{\max}$，使用修订后的 GCL 算法中的两个组成算法可以得到期望竞争比为 2.1839。

我们还证明了三个不可近似结果，包括在至少三台机器存在的情况下，任何随机算法的期望竞争比的下界为 1.6682，对于 $P2|\text{online},t_j,0 \leq p_j \leq u_j|C_{\max}$ 的任何随机算法的期望竞争比的下界为 1.6522，以及对于 $P2|\text{online},t_j,0 \leq p_j \leq u_j|C_{\max}$ 的任何确定性算法的竞争比的下界为 2.2117。根据最后一个下界，我们得出结论，算法 GCL 在期望竞争比方面优于任何确定性算法。这种算法结果在文献中很少见。

算法 GCL 的期望竞争比远高于两台、三台和 $m \geq 4$ 台机器情况下的下界 1.6522、1.6682 和 $2-\frac{1}{m}$，这表明未来的研究可以缩小这些差距，即使是在两台机器的情况下也是如此。可以看到，算法 GCL 中的任务比率测试概率函数在 φ 处不连续，这暗示可能存在一个更好的概率分布函数来选择组成算法。

6 外文翻译参考文献

[1] DÜRR C, ERLEBACH T, MEGOW N, et al. Scheduling with explorable uncertainty[C]//9th Innovations in Theoretical Computer Science Conference (ITCS 2018). 2018.

[2] DÜRR C, ERLEBACH T, MEGOW N, et al. An adversarial model for scheduling with testing[J]. Algorithmica, 2020, 82(12): 3630-3675.

[3] ALBERS S, ECKL A. Explorable uncertainty in scheduling with non-uniform testing times[C]//Approximation and Online Algorithms: 18th International Workshop, WAOA 2020, Virtual Event, September 9–10, 2020, Revised Selected Papers 18. 2021: 127-142.

[4] ALBERS S, ECKL A. Scheduling with testing on multiple identical parallel machines[C]//Algorithms and Data Structures: 17th International Symposium, WADS 2021, Virtual Event, August 9–11, 2021, Proceedings 17. 2021: 29-42.

[5] GAREY MICHAEL R, JOHNSON DAVID S. Computers and Intractability: A guide to the theory of NP-completeness[Z]. 1979.

[6] YAO A C C. Probabilistic computations: Toward a unified measure of complexity[C]//18th Annual Symposium on Foundations of Computer Science (sfcs 1977). 1977: 222-227.

[7] GRAHAM R L. Bounds for certain multiprocessing anomalies[J]. Bell system technical journal, 1966, 45(9): 1563-1581.

[8] BARTAL Y, FIAT A, KARLOFF H, et al. New algorithms for an ancient scheduling problem[C]//Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. 1992: 51-58.

[9] SEIDEN S S. Online randomized multiprocessor scheduling[J]. Algorithmica, 2000, 28(2): 173-216.

[10] ALBERS S. On randomized online scheduling[C]//Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. 2002: 134-143.

四、外文原文

Randomized algorithms for fully online multiprocessor scheduling with testing

Mingyang Gong* Zhi-Zhong Chen† Guohui Lin*‡

May 3, 2023

Abstract

We contribute *the first* randomized algorithm that is an integration of *arbitrarily many* deterministic algorithms for the fully online multiprocessor scheduling with testing problem. When there are only two machines, we show that with two component algorithms its expected competitive ratio is already strictly smaller than the best proven deterministic competitive ratio lower bound. Such algorithmic results are rarely seen in the literature.

Multiprocessor scheduling is one of the first combinatorial optimization problems that have received numerous studies. Recently, several research groups examined its testing variant, in which each job J_j arrives with an upper bound u_j on the processing time and a testing operation of length t_j; one can choose to execute J_j for u_j time, or to test J_j for t_j time to obtain the exact processing time p_j followed by immediately executing the job for p_j time. Our target problem is the fully online multiprocessor scheduling with testing, in which the jobs arrive in sequence so that the testing decision needs to be made at the job arrival as well as the designated machine. We first use Yao's principle to prove lower bounds of 1.6682 and 1.6522 on the expected competitive ratio for any randomized algorithm at the presence of at least three machines and only two machines, respectively, and then propose an expected $(\sqrt{\varphi+3}+1)(\approx 3.1490)$-competitive randomized algorithm as a *non-uniform* probability distribution over arbitrarily many deterministic algorithms, where $\varphi = \frac{\sqrt{5}+1}{2}$ is the Golden ratio. When there are only two machines, we show that our randomized algorithm based on two deterministic algorithms is already expected $\frac{3\varphi+3\sqrt{13-7\varphi}}{4}(\approx 2.1839)$-competitive, while proving a lower bound of 2.2117 on the competitive ratio for any deterministic algorithm.

Keywords: Scheduling; multiprocessor scheduling; scheduling with testing; makespan; randomized algorithm

Acknowledgments.

This research is supported by the NSERC Canada and the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 18K11183.

*Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada. {mgong4, guohui}@ualberta.ca

†Division of Information System Design, Tokyo Denki University. Saitama 350-0394, Japan. zzchen@mail.dendai.ac.jp

‡Correspondence author. Email: guohui@ualberta.ca

1 Introduction

We study the *fully online multiprocessor scheduling with testing* [6, 7, 2, 3] problem in this paper, and study it from the randomized algorithm perspective. Multiprocessor scheduling [10] is one of the first well-known NP-hard combinatorial optimization problems, having received extensive research in the past several decades.

An instance I of multiprocessor scheduling consists of a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, each to be executed *non-preemptively* on one of a set of m parallel identical machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$; and the goal is to minimize the makespan $C_{\max}$, that is, the maximum job completion time. Different from the classic setting where each job J_j comes with the processing time p_j, in *scheduling with testing* each job J_j arrives with an upper bound u_j on the processing time p_j and a testing operation of length t_j, but p_j remains unknown until the job is tested. The job J_j can either be executed on one of the machines for u_j time or be tested for t_j time followed by immediately executing for p_j time on the same machine.

If all the jobs arrive at time zero, multiprocessor scheduling with testing is a *semi-online* problem, denoted as $P \mid t_j, 0 \le p_j \le u_j \mid C_{\max}$. In this paper, we investigate the *fully online* problem in which the jobs arrive in sequence such that the testing decision needs to be made at the job arrival as well as the designated machine for testing and/or executing, denoted as $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$. Apparently, semi-online is a special case of fully online, and in both cases the scheduler should take advantage of the known information about a job upon its arrival to decide whether or not to test the job so as to best balance the total time spent on the job due to the unknown processing time.

Given a polynomial time deterministic algorithm for the semi-online or fully online problem, let $C(I)$ be the makespan produced by the algorithm on an instance I and $C^*(I)$ be the makespan of the optimal *offline* schedule, respectively. The performance of the algorithm is measured by the *competitive ratio* defined as $\sup_I\{C(I)/C^*(I)\}$, where I runs over all instances of the problem, and the algorithm is said $\sup_I\{C(I)/C^*(I)\}$-*competitive*. Switching to a randomized algorithm, we correspondingly collect its *expected* makespan $E[C(I)]$ on the instance I and the randomized algorithm is said *expected* $\sup_I\{E[C(I)]/C^*(I)\}$-competitive. For online problems, randomized algorithms sometimes can better deal with uncertainties leading to lower expected competitive ratios than the competitive ratio of the best deterministic algorithm. We contribute such a randomized algorithm for $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$, and furthermore, when there are only two machines, we show that its expected competitive ratio is strictly smaller than the best proven competitive ratio lower bound of any deterministic algorithm.

We remind the readers that in our problem the job processing is non-preemptive. In the literature, researchers have also considered *preemptive* job processing [6, 7, 2, 3], where any testing or execution operation can be interrupted and resumed later, or the more restricted *test-preemptive* variant [3], where the testing and execution operations of a tested job are non-preemptive but the execution operation does not have to follow immediately the testing operation or on the same machine. Also, our goal is to minimize the makespan $C_{\max}$, that is, the min-max objective; while another important goal to minimize the total job completion time, or the min-sum objective, has received much research too [6, 7, 3].

1.1 Previous work on fully online case

There are not too many existing, deterministic or randomized, approximation algorithms for the fully online problem $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$. We first distinguish a special case where all the testing operations have a unit time, i.e., $t_j = 1$ for every job J_j, called the *uniform testing case* [6, 7, 3], denoted as $P \mid online, t_j = 1, 0 \le p_j \le u_j \mid C_{\max}$. Note that in the *general testing case*, the testing times can be any non-negative values.

When there is only a single machine, the job processing order on the machine is irrelevant to the makespan. This hints that the fully online and the semi-online problems are the same. The first set of results is on the semi-online uniform testing problem $P1 \mid t_j = 1, 0 \le p_j \le u_j \mid C_{\max}$, due to Dürr et al. [6, 7]. They proposed to test the job J_j if $u_j \ge \varphi = \frac{\sqrt{5}+1}{2}$, or to test it with probability $f(u_j) = \max\left\{0, \frac{u_j(u_j-1)}{u_j(u_j-1)+1}\right\}$, leading to a deterministic φ-competitive algorithm and a randomized expected $\frac{4}{3}$-competitive algorithm, respectively. Let $r_j = \frac{u_j}{t_j}$; Albers and Eckl [2] extended the above two algorithms for the general testing case $P1 \mid t_j, 0 \le p_j \le u_j \mid C_{\max}$, to test the job J_j if $r_j \ge \varphi$ or to test it with probability $f(r_j)$, and achieved the same competitive ratio and expected competitive ratio, respectively. The authors [6, 7, 2] showed that both

algorithms are the best possible, that is, φ is a lower bound on the competitive ratio for any deterministic algorithm and $\frac{4}{3}$ is a lower bound on the expected competitive ratio for any randomized algorithm by Yao's principle [17], respectively.

When there are at least two machines, fully online is more general than semi-online. Albers and Eckl [3] proposed to test the job J_j if $r_j \geq \varphi$ in the list scheduling rule [13] that assigns each job to the least loaded machine for processing (possible testing and then executing). They showed that such an algorithm is $\varphi(2-\frac{1}{m})$-competitive and the analysis is tight, where m is the number of machines. They also showed a lower bound of 2 on the competitive ratio for any deterministic algorithm even in the uniform testing case $P \mid online, t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$ [3], and a slightly better lower bound of 2.0953 for the two-machine general testing case $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, i.e., when there are only two machines [3].

1.2 Previous work on semi-online case

See the above reviewed results when there is only a single machine.

When there are at least two machines, recall that in the semi-online problems, all jobs arrive at time zero, and therefore the scheduler can take advantage of all the known u_j and t_j values at time zero for not only job testing decision making but also to form certain job processing orders to better reduce the makespan. Indeed this is the case in the pioneering so-called SBS algorithm by Albers and Eckl [3] for the general testing case, which is 3.1016-competitive (when m tends to infinity). For the uniform testing case, Albers and Eckl [3] also proposed a 3-competitive algorithm, along with a lower bound of $\max\{\varphi, 2-\frac{1}{m}\}$ on the competitive ratio for any deterministic algorithm.

The above two competitive ratios for the general and the uniform testing cases had been improved to 2.9513 and 2.8081 by Gong and Lin [12], respectively, and the state-of-the-art ratios are 2.8019 and 2.5276 by Gong et al. [11], respectively.

For proving the lower bound of $2-\frac{1}{m}$ on the competitive ratio, Albers and Eckl [3] presented an instance of $P \mid t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$ that forces any deterministic algorithm to test all the jobs. This implies that the lower bound is not only on the competitive ratio for any deterministic algorithm, but also on the expected competitive ratio for any randomized algorithm by Yao's principle [17]. Furthermore, the lower bound of $2-\frac{1}{m}$ is then on the expected competitive ratio for any randomized algorithm for the fully online uniform testing problem $P \mid online, t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$.

All the above reviewed competitive ratios and lower bounds are plotted in Figure 1. We point out that for both the fully online and the semi-online problems at the presence of at least two machines, there is no existing randomized algorithm, except deterministic algorithms *trivially* treated as randomized.

1.3 Other related work

We will not review the existing deterministic algorithms for the min-sum objective, but would like to point out that Dürr et al. [6, 7] presented an expected 1.7453-competitive randomized algorithm for the single machine uniform testing problem $P1 \mid t_j = 1, 0 \leq p_j \leq u_j \mid \sum_j C_j$, which tests the to-be-tested jobs in a uniform random order. They also proved a lower bound of 1.6257 on the expected competitive ratio. Albers and Eckl [2] designed a randomized algorithm for the general testing case $P1 \mid t_j, 0 \leq p_j \leq u_j \mid \sum_j C_j$, in which the job J_j is tested with a rather complex probability function in the ratio r_j; the algorithm is expected 3.3794-competitive.

We next review the state-of-the-art randomized algorithms for the classic online multiprocessor scheduling problem $P \mid online \mid C_{\max}$, in which the jobs arrive in sequence, the processing time p_j is revealed when the job J_j arrives, and upon arrival the job must be assigned to a machine irrevocably for execution in order to minimize the makespan. It is well-known that for this problem, the improvement from the competitive ratio 2, by the first deterministic algorithm list-scheduling [13], to the current best 1.9201 [8] took a long time.

For every online approximation algorithm, one of the most challenging parts in performance analysis is to precisely estimate the optimal offline makespan. The three most typical lower bounds are $\frac{1}{m}\sum_{j=1}^{n} p_j$ (the average machine load), $\max_{j=1}^{n} p_j$ (the largest job processing time), and $p_{[m]} + p_{[m+1]}$ where $p_{[j]}$ is the j-th largest processing time among all the jobs. Albers [1] proved that if only the above three lower bounds are used, then the competitive ratio of any deterministic algorithm cannot be smaller than 1.919 when m tends to infinity.

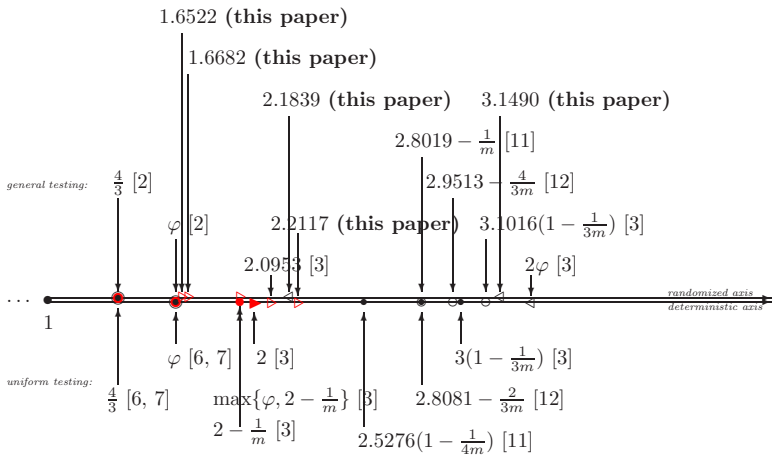

Figure 1: Competitive ratios of the existing deterministic and randomized algorithms for the fully online and semi-online multiprocessor scheduling with testing problems. On the x-axes, the red-colored symbols indicate lower bounds; the triangles ($\triangleleft$'s and $\triangleright$'s for general testing and $\blacktriangleright$ for uniform testing) refer to the results for the fully online problems, while the circles ($\circ$'s for general testing and $\bullet$'s for uniform testing) refer to those for the semi-online problems. Between the two x-axes, the lower one is for deterministic algorithms, while the upper is for randomized ones; the results above the axes are for the general testing case, while those below are for the uniform testing case.

On the positive side, Albers [1] presented an expected 1.916-competitive randomized algorithm, and in the performance analysis only the above three lower bounds on the optimal offline makespan are used. In more details, Albers proposed two deterministic algorithms A_0 and A_1, and proposed to execute each with probability 0.5. The author showed that if the expected makespan is too large compared to the three lower bounds, that is, both A_0 and A_1 produce large makespans, then there are many large jobs present in the instance so that the optimal offline makespan cannot be too small.

Such a randomized algorithm that is a distribution of a constant number of deterministic algorithms is referred to as a *barely randomized algorithm* [14], which takes advantage of its good component algorithms, in that it performs well if at least one component algorithm performs well, and if no component algorithm performs well then the optimal offline makespan is also far away from the lower bounds. This design idea has been used in approximating many other optimization problems and achieved breakthrough results, for example, other scheduling problems [16, 9] and the k-server problem [4]. In fact, for the online multiprocessor scheduling problem $P \mid online \mid C_{\max}$, prior to [1], Seiden [16] modified the *non-barely* randomized algorithms of Bartal et al. [5] and Seiden [15], both of which assign the current job to one of the two least loaded machines with certain probability, to be barely randomized algorithms that are a *uniform* distribution of k deterministic algorithms, where k is the number of schedules to be created inside the algorithm and was chosen big enough to guarantee the expected competitive ratios. While succeeding for small m (specifically, for $m \leq 7$), Albers [1] observed that the analysis of the algorithms in [5, 15, 16] does not work for general large m, and subsequently proposed the new barely randomized algorithm based on only two deterministic algorithms.

1.4 Our results

For the fully online and semi-online multiprocessor scheduling with testing to minimize the makespan, we have seen that the only existing randomized algorithms are the optimal expected $\frac{4}{3}$-competitive algorithms for the case of a single machine [6, 7, 2]. Their expected competitive ratios are strictly smaller than the lower

bound of φ on the competitive ratio for any deterministic algorithm. For the classic online multiprocessor scheduling, only when $2 \leq m \leq 5$, the expected competitive ratios of the algorithms by Bartal et al. [5] and Seiden [15] are strictly smaller than the corresponding lower bounds on the competitive ratio for any deterministic algorithm; the expected competitive ratio 1.916 of the algorithm by Albers [1] is also strictly smaller than the deterministic lower bound 1.919, but *conditional* on using only the three offline makespan lower bounds.

In this paper, we always assume the presence of at least two machines in the fully online problem $P \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$. We propose a barely randomized algorithm that is an integration of *an arbitrary number* of component deterministic algorithms, each is run with certain probability, and show that the expected competitive ratio is at most $\left(\sqrt{(1-\frac{1}{m})^2\varphi^2 + 2(1-\frac{1}{m})} + 1\right)$, where m is the number of machines. This expected competitive ratio is strictly increasing in m, is strictly smaller than the competitive ratio $\varphi(2-\frac{1}{m})$ of the best known deterministic algorithm by Albers and Eckl [3], and approaches $(\sqrt{\varphi+3}+1) \approx 3.1490$ when m tends to infinity. To the best of our knowledge, barely randomized algorithms in the literature are mostly uniform distributions of its component deterministic algorithms, while ours is the *first* non-uniform distribution of its *arbitrary many* component algorithms.

When there are only two machines, that is, for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, we show that employing two component deterministic algorithms in the randomized algorithm leads to an expected competitive ratio of $\frac{3\varphi+3\sqrt{13-7\varphi}}{4} \approx 2.1839$. For this two-machine case, Albers and Eckl [3] proved a lower bound of 2.0953 on the competitive ratio for any deterministic algorithm. We improve it to 2.2117, thus showing that our randomized algorithm beats any deterministic algorithm in terms of the expected competitive ratio. Such algorithmic results are rarely seen in the literature, except those we reviewed in the above.

On the inapproximability, we prove a lower bound of 1.6682 on the expected competitive ratio of any randomized algorithm for $P \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$ at the presence of at least three machines, and a lower bound of 1.6522 on the expected competitive ratio for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, both using a slightly revised Yao's principle [17]. These two lower bounds are strictly better than the lower bound of $2-\frac{1}{m}$ by Albers and Eckl [3] at the presence of three and two machines, respectively. All our results are plotted in Figure 1 as well, highlighted with "this paper".

The rest of the paper is organized as follows. In Section 2, we introduce some basic notations and definitions. Next, in Section 3, we prove the expected competitive ratio lower bounds of 1.6682 and 1.6522 for the fully online problem $P \mid t_j, 0 \leq p_j \leq u_j \mid C_{\max}$ at the presence of at least three machines and only two machines, respectively. Section 4 contains the randomized algorithm for the problem and its performance analysis, the slightly modified randomized algorithm for the two-machine case and its performance analysis, and the lower bound of 2.2117 on the competitive ratio for any deterministic algorithm for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$. Lastly, we conclude the paper in Section 5.

2 Preliminaries

We study the fully online multiprocessor scheduling with testing to minimize the makespan, denoted as $P \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, in which the jobs arrive in sequence. Our goal is to design randomized algorithms with expected competitive ratios that are better than the competitive ratios of the state-of-the-art deterministic algorithms, or even further better than the best proven deterministic competitive ratio lower bounds. The special case where the jobs all arrive at time zero is called semi-online and denoted as $P \mid t_j, 0 \leq p_j \leq u_j \mid C_{\max}$. When applicable, we will prove lower bounds for the semi-online variant.

An instance I of multiprocessor scheduling with testing consists of a job set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, each to be executed on one of a set of m parallel identical machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, where n and $m \geq 2$ are part of the input. Each job J_j arrives with a known upper bound u_j on its processing time p_j and a testing operation of length t_j. The p_j remains unknown until the testing operation is executed. That is, the scheduler can choose not to test the job J_j but execute it for u_j time, or to test it for t_j time followed by immediately execute it on the same machine for p_j time.

A deterministic algorithm A makes a binary decision on whether or not to test a job. Let p_j^A and ρ_j denote the total time spent on the job J_j in the algorithm A and in the optimal offline schedule, respectively. One sees that $p_j^A = t_j + p_j$ if the job is tested or otherwise $p_j^A = u_j$, and $\rho_j = \min\{u_j, t_j + p_j\}$. Let C_j denote the *completion time* of J_j in the schedule generated by the algorithm A. The objective of the problem is

to minimize the makespan $C_{\max} = \max_j C_j$. We use $C^A(I)$ (or C^A when the instance I is clear from the context) and $C^*(I)$ (or C^*, correspondingly) to denote the makespan by the algorithm A and of the optimal offline schedule for the instance I, respectively. Switching to a more general randomized algorithm A, its expected makespan on the instance I is denoted by $E[C^A(I)]$.

Definition 1 (Competitive ratio) *For a deterministic algorithm A, the competitive ratio is the worst-case ratio between the makespan of the schedule produced by the algorithm A and of the optimal offline schedule, that is,* $\sup_I\{C^A(I)/C^*(I)\}$ *where I runs over all instances of the problem.*

For a more general randomized algorithm A, its expected competitive ratio is $\sup_I\{E[C^A(I)]/C^*(I)\}$.

Given that the processing time p_j for the job J_j could go to two extremes of 0 and u_j, a deterministic algorithm often sets up a threshold function on the ratio $r_j = \frac{u_j}{t_j}$ for binary testing decision making. Intuitively, a randomized algorithm should tests the job J_j with probability $f(r_j)$, and such a probability function should be increasing in the ratio r_j and $f(r_j) = 0$ for all $r_j \leq 1$. Indeed, Dürr et al. [6, 7] and Albers and Eckl [2] used the following probability function

$$f(r) = \begin{cases} 0, & \text{if } r \leq 1, \\ \frac{r(r-1)}{r(r-1)+1}, & \text{if } r > 1 \end{cases} \tag{1}$$

in their optimal expected $\frac{4}{3}$-competitive randomized algorithms for the single machine problems $P1 \mid online, t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$ and $P1 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, respectively. Their success unfortunately cannot be easily extended to multiple machines, since one key fact used in their performance analysis is that, for a single machine, the expected makespan equals to the sum of the expected processing times of all the jobs. We show below that, for multiple machines, if a randomized algorithm tests jobs using the probability function in Eq. (1), then its expected competitive ratio is unbounded when the number of machines tends to infinity.

Lemma 1 *For $P \mid online, t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$, if a randomized algorithm tests jobs using the probability function in Eq. (1), then its expected competitive ratio is unbounded when the number of machines tends to infinity.*

Proof. Consider an instance consisting of $m = k(k-1)+1$ machines and $n = k(k-1)+1$ jobs, where k is a positive integer. For each job J_j, $u_j = k$, $t_j = 1$ and $p_j = 0$.

Since $\rho_j = \min\{u_j, t_j + p_j\} = 1$ for each job J_j, in the optimal offline schedule, J_j is tested and scheduled on the machine M_j, leading to a makespan of 1.

The randomized algorithm tests each job with the same probability $\frac{k(k-1)}{k(k-1)+1}$. So the probability that at least one job is untested in the algorithm is

$$1 - \left(\frac{k(k-1)}{k(k-1)+1}\right)^{k(k-1)+1} = 1 - \left(1 - \frac{1}{k(k-1)+1}\right)^{k(k-1)+1}.$$

Since the processing time of an untested job is k, the makespan is at least k. It follows that the expected competitive ratio of the randomized algorithm on this instance is at least

$$k - k\left(1 - \frac{1}{k(k-1)+1}\right)^{k(k-1)+1}.$$

When $m = k(k-1)+1$ tends to infinity, the above becomes $(1 - \frac{1}{e})k$ and approaches infinity too. □

The proof of Lemma 1 suggests to some extent that, when the ratio r_j of J_j is large enough, one should test the job instead of leaving even only a tiny fraction of probability for untesting. Indeed, later we will see that in our proposed randomized algorithm, there is a threshold on r_j's for absolutely testing jobs.

On the other hand, at the presence of at least two machines, we are no longer sure which jobs are assigned to each machine due to random job testing decisions if using a job testing probability function as in Eq. (1). Therefore, we discard the randomized algorithm design idea in [6, 7, 2], but *extend* the idea of Albers [1]

to design *multiple* component deterministic algorithms $A_0, A_1, \ldots, A_\ell$ that complement each other, where ℓ can be made arbitrarily large, and then to run each A_i with certain probability α_i. Let $p_j^{A_i}$ denote the total time spent on the job J_j in A_i, for any $i = 0, 1, \ldots, \ell$. The expected processing time of J_j in the randomized algorithm A is

$$E[p_j^A] = \sum_{i=0}^{\ell} \alpha_i p_j^{A_i}. \tag{2}$$

Similarly, let C^{A_i} be the makespan of the schedule produced by A_i, for any $i = 0, 1, \ldots, \ell$. The expected makespan is

$$E[C^A] = \sum_{i=0}^{\ell} \alpha_i C^{A_i}. \tag{3}$$

The following four lemmas hold for any component deterministic algorithms $A_0, A_1, \ldots, A_\ell$ run with the probability distribution $(\alpha_0, \alpha_1, \ldots, \alpha_\ell)$, while assuming without loss of generality none of them tests any job J_j with $r_j \le 1$.

Lemma 2 *For any $i = 0, 1, \ldots, \ell$, if J_j with $r_j > 1$ is tested in A_i, then $p_j^{A_i} \le (1 + \frac{1}{r_j})\rho_j$.*

PROOF. Note that $p_j^{A_i} = t_j + p_j$. If $\rho_j = t_j + p_j$, then $p_j^{A_i} = \rho_j$ and the lemma is proved. Otherwise, $\rho_j = u_j$; then from $p_j \le u_j$ and $u_j = r_j t_j$, we have $p_j^{A_i} = t_j + p_j \le (\frac{1}{r_j} + 1)u_j = (\frac{1}{r_j} + 1)\rho_j$ and the lemma is also proved. □

Lemma 3 *For any $i = 0, 1, \ldots, \ell$, if J_j with $r_j > 1$ is untested in A_i, then $p_j^{A_i} \le r_j \rho_j$.*

PROOF. Note that $p_j^{A_i} = u_j$. If $\rho_j = u_j$, then $p_j^{A_i} = \rho_j < r_j \rho_j$ and the lemma is proved. Otherwise, $\rho_j = t_j + p_j$; then from $p_j \ge 0$ and $u_j = r_j t_j$, we have $p_j^{A_i} = r_j t_j \le r_j(t_j + p_j) = r_j \rho_j$ and the lemma is also proved. □

Lemma 4 *If J_j with $r_j > 1$ is tested in a subset $\mathcal{T}$ of the algorithms but untested in any of $\{A_0, A_1, \ldots, A_\ell\} \setminus \mathcal{T}$, then*

$$E[p_j^A] \le \max\left\{\theta + (1-\theta)r_j, 1 + \frac{\theta}{r_j}\right\}\rho_j, \text{ where } \theta = \sum_{A_i \in \mathcal{T}} \alpha_i.$$

PROOF. By Eq. (2), $E[p_j^A] = \theta(t_j + p_j) + (1-\theta)u_j$.

If $\rho_j = t_j + p_j$, then $p_j^{A_i} = \rho_j$ for any $A_i \in \mathcal{T}$, and by Lemma 3 $p_j^{A_i} \le r_j \rho_j$ for any $A_i \notin \mathcal{T}$; therefore, $E[p_j^A] \le (\theta + (1-\theta)r_j)\rho_j$. Otherwise, $\rho_j = u_j$; then $p_j^{A_i} = \rho_j$ for any $A_i \notin \mathcal{T}$, and by Lemma 2 $p_j^{A_i} \le (1 + \frac{1}{r_j})\rho_j$ for any $A_i \in \mathcal{T}$, and therefore $E[p_j^A] \le \theta(1 + \frac{1}{r_j})\rho_j + (1-\theta)\rho_j = (1 + \frac{\theta}{r_j})\rho_j$. This proves the lemma. □

Lemma 5 [1] *The following two lower bounds on the optimal offline makespan C^* hold:*

$$C^* \ge \max\left\{\frac{1}{m}\sum_{j=1}^{n} \rho_j, \max_{j=1}^{n} \rho_j\right\}. \tag{4}$$

In the fully online problem $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$, the jobs arrive in sequence and it is assumed to be $\langle J_1, J_2, \ldots, J_n \rangle$. In the semi-online problem $P \mid t_j, 0 \le p_j \le u_j \mid C_{\max}$, all the jobs arrive at time zero and an algorithm can take advantage of all the known u_j's and t_j's to process them in any order.

3 Lower bounds on expected competitive ratios

In this section, we show lower bounds of 1.6682 and 1.6522 on the expected competitive ratio of any randomized algorithm for the fully online problem $P \mid t_j, 0 \leq p_j \leq u_j \mid C_{\max}$ at the presence of at least three machines and only two machines, respectively.

Yao's principle [17] is an efficient tool in proving the lower bounds for randomized algorithms. Below we prove a theorem which is a slight modification of Yao's principle. Let $\mathcal{I}$ and $\mathcal{A}$ be the deterministic instance space and the deterministic algorithm space, respectively, for an optimization problem. For distinction, let I and I_r denote a deterministic instance and a randomized instance, respectively, and let A and A_r denote a deterministic algorithm and a randomized algorithm, respectively. Note that in a randomized instance I_r of the multiprocessor scheduling with testing problem, each job J_j is given known u_j and t_j values and, after the testing is done, a probability distribution of p_j value.

Theorem 1 *For any randomized algorithm A_r and any randomized instance I_r, the expected competitive ratio of A_r satisfies*

$$\sup_{I\in\mathcal{I}} \frac{E[C^{A_r}(I)]}{C^*(I)} \geq \inf_{A\in\mathcal{A}} \frac{E[C^A(I_r)]}{E[C^*(I_r)]},$$

where $E[C^A(I_r)]$ and $E[C^*(I_r)]$ are the expected objective value for a deterministic algorithm A and the expected optimal objective value on the instance I_r, respectively.

Proof. For any deterministic algorithm A, we have

$$E[C^A(I_r)] = \sum_{I\in\mathcal{I}} Pr[I_r = I]C^A(I),$$

where $Pr[I_r = I]$ is the probability that the randomized instance I_r happens to be the deterministic instance I. Note that $\sum_{A\in\mathcal{A}} Pr[A_r = A] = 1$ and $E[C^{A_r}(I)] = \sum_{A\in\mathcal{A}} Pr[A_r = A]C^A(I)$. It follows that

$$\begin{aligned}
\inf_{A\in\mathcal{A}} E[C^A(I_r)] &= \inf_{A\in\mathcal{A}} \sum_{I\in\mathcal{I}} Pr[I_r = I]C^A(I) \\
&\leq \sum_{A\in\mathcal{A}} Pr[A_r = A] \sum_{I\in\mathcal{I}} Pr[I_r = I]C^A(I) \\
&= \sum_{I\in\mathcal{I}} Pr[I_r = I] \sum_{A\in\mathcal{A}} Pr[A_r = A]C^A(I) \\
&= \sum_{I\in\mathcal{I}} Pr[I_r = I]E[C^{A_r}(I)].
\end{aligned}$$

Since $E[C^*(I_r)] = \sum_{I\in\mathcal{I}} Pr[I_r = I]C^*(I)$, at the end we obtain

$$\inf_{A\in\mathcal{A}} \frac{E[C^A(I_r)]}{E[C^*(I_r)]} \leq \frac{\sum_{I\in\mathcal{I}} Pr[I_r = I]E[C^{A_r}(I)]}{\sum_{I\in\mathcal{I}} Pr[I_r = I]C^*(I)} \leq \sup_{I\in\mathcal{I}} \frac{E[C^{A_r}(I)]}{C^*(I)},$$

which proves the theorem. □

For the semi-online single machine uniform testing problem $P1 \mid t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$, the following randomized instance shows the tight lower bound of $\frac{4}{3}$ on expected competitive ratios [6, 7, 2].

Instance 1 *In this randomized instance of $P1 \mid t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$, there is only a single job J_1 with $u_1 = 2$ and $t_1 = 1$, and $p_1 = 0$ or 2 each of probability 0.5.*

In any deterministic algorithm A, J_1 is either tested or untested. If J_1 is untested, then $p_1^A = u_1 = 2$; otherwise J_1 is tested and thus $p_1^A = t_1 + p_1$ which is 1 or 3 each of probability 0.5. It follows that the expected total execution time is $E[p_1^A] = 2$, disregarding J_1 is tested or not.

On the other hand, the optimal offline execution time is $\rho_1 = 1$ if $p_1 = 0$, or 2 if $p_1 = 2$; that is, the expected optimal execution time is $E[\rho_1] = 1.5$. By Theorem 1, $\frac{E[p_1^A]}{E[\rho_1]} = \frac{4}{3}$ is a lower bound on expected competitive ratios for the semi-online single machine uniform testing problem $P1 \mid t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$.

When there are multiple machines, the following deterministic instance shows a lower bound of $2-\frac{1}{m}$ on expected competitive ratios [3].

Instance 2 *In this deterministic instance of $P \mid t_j = 1, 0 \le p_j \le u_j \mid C_{\max}$, every job has $u_i = M$ and $t_i = 1$, where M is sufficiently large, there are $m(m-1)$ small jobs with $p_i = 0$, and additionally a large job with its $p_i = m-1$.*

The optimal offline schedule tests all the jobs, giving rise to $C^* = m$.

On the other hand, the sufficiently large M forces any deterministic algorithm to also test all the jobs. Noting that all these jobs are indistinguishable at the arrival. The adversary decides the processing time of a particular job to be $m-1$, which is the first job assigned by the algorithm to a machine that has a load of $m-1$. This way, the makespan of the generated schedule is $C_{\max} \ge 2m-1$.

For the randomized instance consisting of only this deterministic instance, Theorem 1 implies that $2-\frac{1}{m}$ is a lower bound on expected competitive ratios for the semi-online multiprocessor uniform testing problem $P \mid t_j = 1, 0 \le p_j \le u_j \mid C_{\max}$.

The above two lower bounds on expected competitive ratios for the most special semi-online uniform testing case hold surely for the fully online general testing case too. In the next two theorems, we prove lower bounds specifically for the fully online general testing problem $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$, using randomized instances constructed out of the following deterministic instance. These two lower bounds are better than $2-\frac{1}{m}$ when $m = 2, 3$.

Instance 3 *In this deterministic instance I of $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$, there are m machines and $m+1$ jobs arriving in sequence $\langle J_1, J_2, \cdots, J_{m+1}\rangle$, with*

$$u_j = 2, t_j = 1, \text{ for any } j = 1, 2, \cdots, m+1.$$

That is, these $m+1$ jobs are indistinguishable at the arrivals. For a deterministic algorithm, the adversary sets the first two untested jobs, if exist, or otherwise the last two jobs in the sequence to have their exact executing times 0; the other $m-1$ jobs have their exact executing times 2.

For example, when $m = 4$ and the algorithm does not test any job, then the sequence of exact executing times is $\langle 0, 0, 2, 2, 2\rangle$; if the algorithm does not test any of J_3, J_4 and J_5, then the sequence of exact executing times is $\langle 2, 2, 0, 0, 2\rangle$; if the algorithm does not test J_1 only, then the sequence of exact executing times is $\langle 0, 2, 2, 2, 0\rangle$; if the algorithm tests all the five jobs, then the sequence of exact executing times is $\langle 2, 2, 2, 0, 0\rangle$.

Given a deterministic algorithm A, let $C^A_{\min}(I)$ denote the minimum machine load in the schedule produced by A for the instance I.

Lemma 6 *When $m \ge 3$, on the instance I in Instance 3, for any deterministic algorithm A, either $C^A(I) \ge 4$, or $C^A(I) = 3$ and $C_{\min}(I) \ge 2$.*

PROOF. If the algorithm A does not test two or more jobs, then one sees that the total processing time of every job is at least 2, leading to the makespan $C^A(I) \ge 4$ by the Pigeonhole Principle.

If the algorithm A tests all but at most one job, then among the first m jobs $\langle J_1, J_2, \cdots, J_m\rangle$, the total processing time of each but one of them is 3, and the total processing time of the exceptional job is either 1 if it is tested or 2 if it is untested. One sees that when a machine is assigned with any two of these m jobs, the makespan $C^A(I) \ge 4$ by the Pigeonhole Principle. In the other case where each machine is assigned with exactly one of these m jobs, assigning the last job J_{m+1} to any one of the machines would lead to either $C^A(I) \ge 4$, or $C^A(I) = 3$ and $C_{\min}(I) \ge 2$. This finishes the proof. □

Lemma 7 *When $m = 2$, on the instance I in Instance 3, for any deterministic algorithm A, either $C^A(I) \ge 5$, or $C^A(I) \ge 4$ and $C_{\min}(I) \ge 1$, or $C^A(I) \ge 3$ and $C_{\min}(I) \ge 2$.*

PROOF. Note that I is now a three-job instance.

We distinguish the two cases on whether A tests the first job J_1 or not. If J_1 is tested, then $p_1 = 2$ and thus $p^A_1 = 3$. Note that the total processing times of J_2 and J_3 are at least 1. These three values $\{3, 1, 1\}$ guarantee one of the makespan scenarios.

If J_1 is untested, then $p_1^A = 2$. Note that the total processing time of one of J_2 and J_3 is at least 2 (which is either untested, or tested with the exact executing time 2), and of the other is at least 1. These three values $\{2, 2, 1\}$ guarantee one of the makespan scenarios. □

Theorem 2 *The expected competitive ratio of any randomized algorithm for $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{max}$ at the presence of three or more machines is at least $\frac{21}{2} - \sqrt{78} \approx 1.6682$.*

Proof. We consider a randomized instance I_r, which is a probability distribution over three deterministic instances I_1, I_2 and I_3. These three instances are all extended from I in Instance 3, by appending a job J_{m+2} at the end of the job sequence. Specifically, for J_{m+2},

$$(u_{m+2}, t_{m+2}, p_{m+2}) = \begin{cases} (0,0,0), & \text{in } I_1, \\ (2+\frac{1}{\alpha}, 3, 0), & \text{in } I_2, \\ (2+\frac{1}{\alpha}, 3, 2+\frac{1}{\alpha}), & \text{in } I_3, \end{cases}$$

where $\alpha = \frac{2\sqrt{78}-5}{41} \approx 0.3089$, and its probability distribution is $(\beta_1, \beta_2, \beta_3) = (\frac{2-\alpha}{3}, \alpha, \frac{1-2\alpha}{3})$.

Note that exactly two of $\rho_1, \rho_2, \cdots, \rho_{m+1}$ are 1, while the others are 2. For the job J_{m+2}, $\rho_{m+2} = 0, 3, 2+\frac{1}{\alpha}$ in I_1, I_2, I_3, respectively. It follows from $m \ge 3$ that the optimal offline makespans of I_1, I_2, I_3 are $C^* = 2, 3, 2+\frac{1}{\alpha}$, respectively. Therefore, the expected optimal offline makespan of I_r is

$$E[C^*(I_r)] = 2\beta_1 + 3\beta_2 + (2+\frac{1}{\alpha})\beta_3 = \frac{4}{3} + \alpha + \frac{1}{3\alpha}.$$

For any deterministic algorithm A, it performs the same on the prefix instance I of all the three instances I_1, I_2 and I_3, including which jobs of the first $m+1$ ones are tested, to which machines they are assigned to, and all the m machine loads when the last job J_{m+2} arrives. In particular, since u_{m+2} and t_{m+2} are the same in I_2 and I_3, J_{m+2} is either both tested or both untested by A. By Lemma 6, we have either $C^A(I_1) = C^A(I) \ge 4$, or $C^A(I_1) = C^A(I) = 3$ and $C^A_{\min}(I_1) = C^A_{\min}(I) \ge 2$. We distinguish the following four cases on $C^A(I)$ and whether J_{m+2} of I_2 is tested by A or not.

The first case is that $C^A(I) \ge 4$ and J_{m+2} of I_2 is tested by A. Clearly, $C^A(I_2) \ge 4$ and $C^A(I_3) \ge p^A_{m+2} = 5 + \frac{1}{\alpha}$. Therefore,

$$E[C^A(I_r)] \ge 4\beta_1 + 4\beta_2 + (5+\frac{1}{\alpha})\beta_3 = \frac{11}{3} - \frac{2\alpha}{3} + \frac{1}{3\alpha}.$$

The second case is that $C^A(I) \ge 4$ and J_{m+2} of I_2 is untested by A. We have $C^A(I_i) \ge p^A_{m+2} = 2 + \frac{1}{\alpha}$ for both $i = 2, 3$. Therefore,

$$E[C^A(I_r)] \ge 4\beta_1 + (2+\frac{1}{\alpha})(\beta_2 + \beta_3) = \frac{11}{3} - \frac{2\alpha}{3} + \frac{1}{3\alpha}.$$

The third case is that $C^A(I) = 3$, $C^A_{\min}(I) \ge 2$, and J_{m+2} of I_2 is tested by A. For I_2, $p^A_{m+2} = 3$ and thus $C^A(I_2) \ge 5$ no matter which machine J_{m+2} is assigned to; for I_3, $p^A_{m+2} = 5 + \frac{1}{\alpha}$ and thus $C^A(I_3) \ge 7 + \frac{1}{\alpha}$. Therefore,

$$E[C^A(I_r)] \ge 3\beta_1 + 5\beta_2 + (7+\frac{1}{\alpha})\beta_3 = \frac{11}{3} - \frac{2\alpha}{3} + \frac{1}{3\alpha}.$$

The last case is that $C^A(I) = 3$, $C_{\min}(I) \ge 2$, and J_{m+2} of I_2 is untested by A. For both I_2 and I_3, $p^A_{m+2} = 2 + \frac{1}{\alpha}$ and thus $C^A(I_i) \ge 4 + \frac{1}{\alpha}$ no matter which machine J_{m+2} is assigned to, for both $i = 2, 3$. Therefore,

$$E[C^A(I_r)] \ge 3\beta_1 + (4+\frac{1}{\alpha})(\beta_2 + \beta_3) = \frac{11}{3} + \frac{\alpha}{3} + \frac{1}{3\alpha} > \frac{11}{3} - \frac{2\alpha}{3} + \frac{1}{3\alpha}.$$

To conclude, by Theorem 1, the expected competitive ratio of any randomized algorithm for $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$ at the presence of three or more machines is at least

$$\inf_A \frac{E[C^A(I_r)]}{E[C^*(I_r)]} \ge \frac{\frac{11}{3} - \frac{2\alpha}{3} + \frac{1}{3\alpha}}{\frac{4}{3} + \alpha + \frac{1}{3\alpha}} = \frac{21}{2} - \sqrt{78}.$$

This completes the proof of the theorem. □

Theorem 3 *The expected competitive ratio of any randomized algorithm for* $P2 \mid online, t_j, 0 \le p_j \le u_j \mid C_{max}$ *is at least* $\frac{21+4\sqrt{51}}{30} \approx 1.6522$.

PROOF. Again we consider a randomized instance I_r, which is a probability distribution over three deterministic instances I_1, I_2 and I_3. These three instances are all extended from the three-job instance I in Instance 3 when $m = 2$, by appending a job J_4 at the end of the job sequence. Specifically, for J_4,

$$(u_4, t_4, p_4) = \begin{cases} (0,0,0), & \text{in } I_1, \\ (\frac{2}{\alpha}, 4, 0), & \text{in } I_2, \\ (\frac{2}{\alpha}, 4, \frac{2}{\alpha}), & \text{in } I_3, \end{cases}$$

where $\alpha = \frac{\sqrt{51}-4}{10} \approx 0.3141$, and its probability distribution is $(\beta_1, \beta_2, \beta_3) = (\frac{1}{2}, \alpha, \frac{1}{2} - \alpha)$.

Recall that two of ρ_1, ρ_2, ρ_3 are 1 while the other is 2. For the job J_4, $\rho_4 = 0, 4, \frac{2}{\alpha}$ in I_1, I_2, I_3, respectively. It follows that the optimal offline makespans of I_1, I_2, I_3 are $C^* = 2, 4, \frac{2}{\alpha}$, respectively. Therefore, the expected optimal offline makespan of I_r is

$$E[C^*(I_r)] = 2\beta_1 + 4\beta_2 + \frac{2}{\alpha}\beta_3 = 4\alpha + \frac{1}{\alpha} - 1.$$

For any deterministic algorithm A, it performs the same on the prefix instance I of all the three instances I_1, I_2 and I_3, including which of the first three jobs are tested, to which machines they are assigned to, and the two machine loads when the last job J_4 arrives. In particular, since u_4 and t_4 are the same in I_2 and I_3, J_4 is either both tested or both untested by A. By Lemma 7, we have either $C^A(I_1) = C^A(I) \ge 5$, or $C^A(I_1) = C^A(I) \ge 4$ and $C^A_{\min}(I_1) = C^A_{\min}(I) \ge 1$, or $C^A(I_1) = C^A(I) \ge 3$ and $C^A_{\min}(I_1) = C^A_{\min}(I) \ge 2$. We distinguish the following six cases on $C^A(I)$ and whether J_4 of I_2 is tested by A or not.

When $C^A(I) \ge 5$ and J_4 of I_2 is tested by A, we have $C^A(I_2) \ge 5$ and $C^A(I_3) \ge p_4^A = 4 + \frac{2}{\alpha}$, leading to

$$E[C^A(I_r)] \ge 5\beta_1 + 5\beta_2 + (4 + \frac{2}{\alpha})\beta_3 = \frac{5}{2} + \alpha + \frac{1}{\alpha} > \frac{5}{2} + \frac{1}{\alpha}.$$

When $C^A(I) \ge 5$ and J_4 of I_2 is untested by A, we have $C^A(I_i) \ge p_4^A = \frac{2}{\alpha}$ for both $i = 2, 3$, leading to

$$E[C^A(I_r)] \ge 5\beta_1 + \frac{2}{\alpha}(\beta_2 + \beta_3) = \frac{5}{2} + \frac{1}{\alpha}.$$

When $C^A(I) \ge 4$ and $C_{\min}(I) \ge 1$, and J_4 of I_2 is tested by A, we have $p_4^A = 4$ in I_2 and thus $C^A(I_2) \ge p_4^A + 1 = 5$, and similarly $C^A(I_3) \ge p_4^A + 1 = 5 + \frac{2}{\alpha}$. They lead to

$$E[C^A(I_r)] \ge 4\beta_1 + 5\beta_2 + (5 + \frac{2}{\alpha})\beta_3 = \frac{5}{2} + \frac{1}{\alpha}.$$

When $C^A(I) \ge 4$ and $C_{\min}(I) \ge 1$, and J_4 of I_2 is untested by A, we have $p_4^A = \frac{2}{\alpha}$ in both I_2 and I_3 and thus $C^A(I_i) \ge p_4^A + 1 = 1 + \frac{2}{\alpha}$ for both $i = 2, 3$. They lead to

$$E[C^A(I_r)] \ge 4\beta_1 + (1 + \frac{2}{\alpha})(\beta_2 + \beta_3) = \frac{5}{2} + \frac{1}{\alpha}.$$

When $C^A(I) \ge 3$ and $C_{\min}(I) \ge 2$, and J_4 of I_2 is tested by A, we have $p_4^A = 4$ in I_2 and thus $C^A(I_2) \ge p_4^A + 2 = 6$, and similarly $C^A(I_3) \ge p_4^A + 2 = 6 + \frac{2}{\alpha}$. They lead to

$$E[C^A(I_r)] \ge 3\beta_1 + 6\beta_2 + (6 + \frac{2}{\alpha})\beta_3 = \frac{5}{2} + \frac{1}{\alpha}.$$

Lastly, when $C^A(I) \ge 3$ and $C_{\min}(I) \ge 2$, and J_4 of I_2 is untested by A, we have $p_4^A = \frac{2}{\alpha}$ in both I_2 and I_3 and thus $C^A(I_i) \ge p_4^A + 2 = 2 + \frac{2}{\alpha}$ for both $i = 2, 3$. They lead to

$$E[C^A(I_r)] \ge 3\beta_1 + (2 + \frac{2}{\alpha})(\beta_2 + \beta_3) = \frac{5}{2} + \frac{1}{\alpha}.$$

To conclude, by Theorem 1, the expected competitive ratio of any randomized algorithm for $P2 \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$ is at least

$$\inf_A \frac{E[C^A(I_r)]}{E[C^*(I_r)]} \ge \frac{\frac{5}{2}+\frac{1}{\alpha}}{4\alpha+\frac{1}{\alpha}-1} = \frac{21+4\sqrt{51}}{30}.$$

This completes the proof of the theorem. □

4 Randomized algorithms

In this section, we first present a randomized algorithm for the fully online multiprocessor scheduling with testing problem $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$ at the presence of at least two machines, and show that its expected competitive ratio is $(\sqrt{\varphi+3}+1) \approx 3.1490$. When there are only two machines, we revise slightly the parameters in the algorithm leading to an improved expected competitive ratio of $\frac{3\varphi+3\sqrt{13-7\varphi}}{4} \approx 2.1839$. We last prove an improved lower bound of 2.2117 on the competitive ratio for any deterministic algorithm for the two machine case, implying that our randomized algorithm is better than any deterministic algorithm in terms of expected competitive ratio.

For the fully online multiprocessor scheduling with testing problem $P \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$, Albers and Eckl [3] proposed to test the job J_j if $r_j \ge \varphi$ in the list scheduling rule [13] that assigns each job to the least loaded machine for processing (possible testing and then executing). They showed that such a deterministic algorithm is tight $\varphi(2-\frac{1}{m})$-competitive, where m is the number of machines. We employ it as our first component algorithm A_0, and it is run with probability $\alpha_0 = \alpha(m,\ell)$ as in Eq. (5), where $\ell \ge 1$ is a fixed constant chosen by our randomized algorithm.

$$\alpha_0 = \alpha(m,\ell) = \sqrt{\frac{(1-\frac{1}{m})(\ell+1)\varphi^2}{(1-\frac{1}{m})(\ell+1)\varphi^2+2\ell}}, \text{ and } \alpha_i = \beta(m,\ell) = \frac{1-\alpha(m,\ell)}{\ell}, \text{ for } i = 1,2,\ldots,\ell. \quad (5)$$

Below we simplify $\alpha(m,\ell)$ and $\beta(m,\ell)$ as α and β, respectively.

One sees that when $r_j \le 1$, the job J_j should not be tested, as this is the case in A_0. On the other hand, when r_j is large, testing does not waste too much time even if p_j is very close to u_j, as this is also the case in A_0. The "risk" of making wrong testing decision happens when r_j is close to φ, in that p_j could be very close to 0 if J_j is untested while be very close to u_j if tested. Nevertheless, we have also seen from the proof of Lemma 1 that even a tiny fraction of probability for untesting jobs with large ratio r_j could lead to an unbounded expected competitive ratio, when the number m of machines tends to infinity. Therefore, in each of the other component algorithms A_i, $i = 1,2,\ldots,\ell$, we set up a threshold $y_i(m,\ell) > \varphi$ as in Eq. (6) such that a job J_j with $r_j \ge y_i(m,\ell)$ is tested in A_i; we then set up another threshold $x_i(m,\ell) = 1+\frac{1}{y_i(m,\ell)}$ such that a job J_j with $r_j \le x_i(m,\ell)$ is untested in A_i.

$$y_i(m,\ell) = \frac{\varphi(\alpha+i\beta)}{\alpha}, \; x_i(m,\ell) = 1+\frac{1}{y_i(m,\ell)}, \text{ for } i = 0,1,\ldots,\ell. \quad (6)$$

Similarly, below we simplify $y_i(m,\ell)$ and $x_i(m,\ell)$ as y_i and x_i, respectively. When $r_j \in (x_i, y_i)$, we invert the testing decision as in A_0 to test J_j if $r_j \le \varphi$ or untest J_j if $r_j > \varphi$ in A_i. Our randomized algorithm, denoted as GCL, chooses to run A_i with probability α_i, for $i = 0,1,\ldots,\ell$.

From Eqs. (5, 6), one sees that $\alpha, \beta > 0$, and

$$1 < x_\ell < \ldots < x_1 < x_0 = \varphi = y_0 < y_1 < \ldots < y_\ell = \frac{\varphi}{\alpha}. \quad (7)$$

A high-level description of the algorithm GCL is presented in Figure 2.

The inequalities in the next lemma will be convenient in the performance analysis for the algorithm GCL.

Lemma 8 $\frac{1-\alpha}{x_\ell} < \frac{\alpha}{\varphi}$, *and* $(\ell-i)\beta(y_{i+1}-1) < \frac{\alpha}{\varphi}$ *for every* $i = 0,1,\ldots,\ell-1$.

Algorithm GCL:
Input: m machines, and n jobs arriving in sequence $\langle J_1, J_2, \ldots, J_n\rangle$;
Output: A schedule in which each job is tested or not and the machine processing the job.

1. Choose ℓ and set up parameters α_i, y_i and x_i as in Eqs. (5, 6), for $i = 0, 1, \ldots, \ell$;
2. Component algorithm A_i, $i = 0, 1, \ldots, \ell$: For each job J_j,
   2.1 if $r_j \le x_i$ or $\varphi < r_j \le y_i$, then assign J_j untested on the least loaded machine for executing;
   2.2 otherwise, assign J_j on the least loaded machine for testing and executing;
3. Choose to run A_i with probability α_i, for $i = 0, 1, \ldots, \ell$.

Figure 2: A high level description of the algorithm GCL.

PROOF. By its definition in Eq. (5), α is strictly increasing in m. Therefore,

$$\sqrt{\frac{(\ell+1)\varphi^2}{(\ell+1)\varphi^2+2\ell}} > \alpha = \alpha(m,\ell) \ge \alpha(2,\ell) = \sqrt{\frac{(\ell+1)\varphi^2}{(\ell+1)\varphi^2+4\ell}} \ge \sqrt{\frac{\varphi^2}{\varphi^2+4}} > \varphi - 1. \tag{8}$$

Using $y_\ell = \frac{\varphi}{\alpha}$ and $x_\ell = 1 + \frac{\alpha}{\varphi}$, we have $x_\ell > 1 + \frac{\varphi-1}{\varphi} = 3 - \varphi$. It follows that $(\varphi + x_\ell)\alpha > 3(\varphi - 1) > \varphi$, and thus

$$\frac{\alpha}{\varphi} - \frac{1-\alpha}{x_\ell} = \frac{(\varphi + x_\ell)\alpha - \varphi}{\varphi x_\ell} > 0,$$

which is the first inequality.

Define $g(i) = (\ell - i)\beta(y_{i+1} - 1)$ for $i = 0, 1, \ldots, \ell - 1$, which is a concave quadratic function in i and its axis of symmetry is $i = i_0 = \frac{1}{2}(\ell - 1 - \frac{\alpha}{(\varphi+1)\beta})$. Note that when $\ell = 1$, Eq. (8) implies $\sqrt{\frac{\varphi^2}{\varphi^2+1}} > \alpha \ge \sqrt{\frac{\varphi^2}{\varphi^2+2}}$; thus by Eq. (6), we have $g(0) = \beta(\frac{\varphi}{\alpha} - 1) = (1-\alpha)(\frac{\varphi}{\alpha} - 1) < \frac{\alpha}{\varphi}$ and the lemma is proved. When $\ell = 2$, Eq. (8) implies $\sqrt{\frac{3\varphi^2}{3\varphi^2+4}} > \alpha \ge \sqrt{\frac{3\varphi^2}{3\varphi^2+8}}$; thus by Eq. (6), we have $g(i) \le 2\beta(\frac{\varphi}{\alpha} - 1) = (1-\alpha)(\frac{\varphi}{\alpha} - 1) < \frac{\alpha}{\varphi}$ for $i = 0, 1$ and the lemma is proved.

Below we assume that $\ell \ge 3$. Note that when $i \le (2-\varphi)\ell - 1$, Eqs. (5, 6, 8) together imply

$$y_{i+1} \le \frac{\varphi(\alpha + (2-\varphi)\ell\beta)}{\alpha} = \frac{\alpha + \varphi - 1}{\alpha} < 2$$

and thus

$$g(i) \le (\ell - i)\beta \le 1 - \alpha < 2 - \varphi = \frac{\varphi - 1}{\varphi} < \frac{\alpha}{\varphi}.$$

Using $\beta = \frac{1}{\ell}(1-\alpha) \le \frac{1}{3}(1-\alpha)$ and Eq. (8), one sees that

$$(2-\varphi)\ell - 1 - i_0 = \frac{\varphi\alpha - (\varphi - 1) - (\varphi + 1)\beta}{2(\varphi+1)\beta} > \frac{5 - 3\varphi}{6(\varphi+1)\beta} > 0,$$

that is, $(2-\varphi)\ell - 1$ is to the right of the axis of symmetry. Therefore, for every $i = 0, 1, \ldots, \ell - 1$, $g(i) < \frac{\alpha}{\varphi}$. This completes the proof of the lemma. □

Lemma 9 *In the algorithm A_i, $p_j^{A_i} \le y_i\rho_j$ for every job J_j, for any $i = 0, 1, \ldots, \ell$.*

PROOF. In the algorithm A_0, a job J_j is untested if and only if $r_j \le \varphi$. By Lemmas 2 and 3, $p_j^{A_0} \le \max\{1 + \frac{1}{\varphi}, \varphi\}\rho_j = y_0\rho_j$.

Consider the algorithm A_i, for any $i = 1, 2, \ldots, \ell$. If $r_j \le x_i$ or $\varphi < r_j \le y_i$, then J_j is untested in A_i and, by Lemma 3, $p_j^{A_i} \le y_i\rho_j$. Otherwise, $x_i < r_j \le \varphi$ or $r_j > y_i$, J_j is tested and, by Lemma 2, $p_j^{A_i} \le (1 + \frac{1}{x_i})\rho_j$. In conclusion, for each job J_j, we have

$$p_j^{A_i} \le \max\left\{y_i, 1 + \frac{1}{x_i}\right\}\rho_j \le y_i\rho_j.$$

This proves the lemma. □

Lemma 10 *In the algorithm* GCL, $E[p_j^A] \leq x_\ell \rho_j$ *for every job* J_j.

PROOF. If $r_j > y_\ell$, then J_j is tested in every algorithm A_i, for $i = 0, 1, \ldots, \ell$. So, by Lemma 4, $E[p_j^A] \leq (1 + \frac{1}{y_\ell})\rho_j = x_\ell \rho_j$. If $r_j \leq x_\ell$, then J_j is untested in every algorithm A_i, $i = 0, 1, \ldots, \ell$. So, by Lemma 4, $E[p_j^A] \leq x_\ell \rho_j$.

If $y_h < r_j \leq y_{h+1}$, for some $h = 0, 1, \ldots, \ell - 1$, then J_j is tested in $A_0, A_1, \ldots, A_h$ but untested in $A_{h+1}, A_{h+2}, \ldots, A_\ell$. Lemma 4 and then Eq. (6) and Lemma 8 together imply that

$$\begin{aligned} E[p_j^A] &\leq \max\left\{\alpha + h\beta + (\ell - h)\beta y_{h+1}, 1 + \frac{\alpha + h\beta}{y_h}\right\}\rho_j \\ &= \max\left\{1 + (\ell - h)\beta(y_{h+1} - 1), 1 + \frac{\alpha}{\varphi}\right\}\rho_j \\ &\leq (1 + \frac{\alpha}{\varphi})\rho_j = x_\ell \rho_j. \end{aligned}$$

Lastly, if $x_{h+1} < r_j \leq x_h$, for some $h = 0, 1, \ldots, \ell - 1$, then J_j is untested in $A_0, A_1, \ldots, A_h$ but tested in $A_{h+1}, A_{h+2}, \ldots, A_\ell$. Lemma 4 and then Eq. (6), Lemma 8 and Eq. (8) together imply that

$$\begin{aligned} E[p_j^A] &\leq \max\left\{(\ell - h)\beta + (1 - (\ell - h)\beta)x_h, 1 + \frac{(\ell - h)\beta}{x_{h+1}}\right\}\rho_j \\ &= \max\left\{1 + \frac{\alpha}{\varphi}, 1 + \frac{(\ell - h)\beta}{x_{h+1}}\right\}\rho_j \\ &\leq \max\left\{1 + \frac{\alpha}{\varphi}, 1 + \frac{1 - \alpha}{x_\ell}\right\}\rho_j \\ &\leq (1 + \frac{\alpha}{\varphi})\rho_j = x_\ell \rho_j. \end{aligned}$$

This proves the lemma. □

Theorem 4 *The expected competitive ratio of* GCL *for the problem* $P \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$ *is at most* $\sqrt{(1 - \frac{1}{m})^2(1 + \frac{1}{\ell})^2\varphi^2 + 2(1 - \frac{1}{m})(1 + \frac{1}{\ell}) + 1} - (1 - \frac{1}{m})\frac{\varphi}{\ell}$ *by setting* $\alpha(m, \ell), \beta(m, \ell)$ *as in Eq. (5) and* $y_i(m, \ell), x_i(m, \ell)$ *for every* $i = 0, 1, \ldots, \ell$ *as in Eq. (6), where* $m \geq 2$ *is the number of machines and* $\ell \geq 1$ *is a fixed constant.*

PROOF. Suppose the job J_{n_i} determines the makespan of the schedule produced by the component algorithm A_i, for each $i = 0, 1, \ldots, \ell$, respectively. Note that J_{n_i} is assigned to the least loaded machine by the algorithm A_i. Therefore, the makespan

$$C^{A_i} \leq \frac{1}{m}\sum_{j=1}^{n_i - 1} p_j^{A_i} + p_{n_i}^{A_i} = \frac{1}{m}\sum_{j=1}^{n_i} p_j^{A_i} + (1 - \frac{1}{m})p_{n_i}^{A_i} \leq \frac{1}{m}\sum_{j=1}^{n} p_j^{A_i} + (1 - \frac{1}{m})p_{n_i}^{A_i}.$$

Using Eqs. (3), (7), Lemmas 9, 10 and the lower bounds in Lemma 5, the expected makespan is at most

$$\begin{aligned}
E[C^A] &= \alpha C^{A_0} + \sum_{i=1}^{\ell} \beta C^{A_i} \\
&\le \frac{1}{m}\sum_{j=1}^{n} E[p_j^A] + \alpha(1-\frac{1}{m})p_{n_0}^{A_0} + \sum_{i=1}^{\ell}\beta(1-\frac{1}{m})p_{n_i}^{A_i} \\
&\le \frac{x_\ell}{m}\sum_{j=1}^{n}\rho_j + \alpha(1-\frac{1}{m})\varphi\rho_{n_0} + \sum_{i=1}^{\ell}\beta(1-\frac{1}{m})y_i\rho_{n_i} \\
&\le \left(x_\ell + (1-\frac{1}{m})\left(\alpha\varphi + \beta\sum_{i=1}^{\ell} y_i\right)\right) C^* \\
&= \left(1+\frac{\alpha}{\varphi} + (1-\frac{1}{m})\left(\alpha\varphi + \ell\beta\varphi + \frac{\ell(\ell+1)\beta^2\varphi}{2\alpha}\right)\right) C^* \\
&= \left(1+\frac{\alpha}{\varphi} + (1-\frac{1}{m})\left(\varphi + \frac{(\ell+1)(1-\alpha)^2\varphi}{2\ell\alpha}\right)\right) C^* \\
&= \left(\left(\frac{1}{\varphi} + (1-\frac{1}{m})\frac{(\ell+1)\varphi}{2\ell}\right)\alpha + (1-\frac{1}{m})\frac{(\ell+1)\varphi}{2\ell}\frac{1}{\alpha} + 1 - (1-\frac{1}{m})\frac{\varphi}{\ell}\right) C^* \\
&= \left(\sqrt{(1-\frac{1}{m})^2(1+\frac{1}{\ell})^2\varphi^2 + 2(1-\frac{1}{m})(1+\frac{1}{\ell})} + 1 - (1-\frac{1}{m})\frac{\varphi}{\ell}\right) C^*,
\end{aligned}$$

and thus the theorem is proved. □

By choosing a sufficiently large ℓ, the expected competitive ratio of the algorithm GCL approaches $\sqrt{(1-\frac{1}{m})^2\varphi^2 + 2(1-\frac{1}{m})} + 1$, which is increasing in m and approaches $\sqrt{\varphi+3}+1 \approx 3.1490$ when m tends to infinity. Correspondingly, one can check that α, y_ℓ and x_ℓ approach $\sqrt{\frac{\varphi+1}{\varphi+3}} \approx 0.7529$, $\sqrt{\varphi+3} \approx 2.1490$ and $1+\frac{1}{\sqrt{\varphi+3}} \approx 1.4653$, respectively.

In the algorithm GCL, we can determine the testing probability $f(r)$ of a job as a function in its ratio r. For example, $f(r) = 1$ if $r \ge y_\ell$ and $f(r) = 0$ if $r \le x_\ell$. With a fixed constant ℓ, one sees that such a probability function $f(r)$ is staircase and increasing. The limit of this probability function when ℓ tends to infinity, still denoted as $f(r)$, is interesting. First, note that $\alpha(m,\ell)$ approaches $\alpha(m) = \sqrt{\frac{(1-\frac{1}{m})\varphi^2}{(1-\frac{1}{m})\varphi^2+2}}$, $y_\ell(m,\ell)$ and $x_\ell(m,\ell)$ approach $y(m) = \frac{\varphi}{\alpha(m)}$ and $x(m) = 1+\frac{1}{y(m)}$, respectively. Then, by Eq.(6), we have

$$f(r) = \begin{cases} 0, & \text{when } r \le x(m), \\ 1 - \frac{\alpha(m)}{\varphi(r-1)}, & \text{when } x(m) < r \le \varphi, \\ \frac{\alpha(m)r}{\varphi}, & \text{when } \varphi < r \le y(m), \\ 1, & \text{when } r > y(m), \end{cases}$$

which is increasing, and continuous everywhere except at φ.

When there are only two machines, i.e., $m = 2$, by choosing a sufficiently large ℓ, the algorithm GCL is already expected $\frac{1}{2}\sqrt{\varphi+5}+1 \approx 2.2863$-competitive for the problem $P2 \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$. Note that inside the algorithm, the probability of choosing the algorithm A_0 approaches $\alpha(2) = \frac{\varphi}{\sqrt{\varphi+5}} \approx 0.6290$. We next show that, by choosing another probability for running A_0, the revised GCL algorithm based on two component algorithms is expected $\frac{3\varphi+3\sqrt{13-7\varphi}}{4} \approx 2.1839$-competitive.

Theorem 5 *The expected competitive ratio of the revised* GCL *algorithm for the problem* $P2 \mid online, t_j, 0 \le p_j \le u_j \mid C_{\max}$ *is at most* $\frac{3\varphi+3\sqrt{13-7\varphi}}{4}$ *by using two component algorithms and by setting*

$$\alpha = \varphi - 1 \approx 0.6180, x_1 = \frac{\varphi + \sqrt{13-7\varphi}}{2} \approx 1.4559, \text{ and } y_1 = \frac{1}{x_1 - 1} \approx 2.1935. \tag{9}$$

Furthermore, the expected competitive ratio is tight.

Proof. The three parameters in Eq. (9) satisfy the algorithm design: $1 < x_1 < \varphi < y_1$ and $\alpha < 1$.

Furthermore, they imply that $1 - \alpha + \alpha\varphi = 3 - \varphi < x_1$ and $1 + \frac{\alpha}{\varphi} = 2 - \frac{1}{\varphi} < x_1$. Also noticing that x_1 is a root to the quadratic equation $x^2 - \varphi x + 2\varphi - 3 = 0$, we have $x_1^2 = \varphi x_1 + 3 - 2\varphi$. It follows that

$$y_1 - 1 - \frac{1}{x_1} = \frac{1 + x_1 - x_1^2}{x_1(x_1 - 1)} = \frac{(\varphi - 1)(2 - x_1)}{x_1(x_1 - 1)} > 0,$$

$$x_1 - 1 - \frac{1 - \alpha}{x_1} = \frac{x_1^2 - x_1 - (2 - \varphi)}{x_1} = \frac{(\varphi - 1)(x_1 - 1)}{x_1} > 0,$$

and

$$x_1 - \alpha - (1 - \alpha)y_1 = \frac{x_1^2 - \varphi x_1 + 2\varphi - 3}{x_1 - 1} = 0.$$

For each job J_j, if $r_j \le x_1$, then it is not tested in any of A_0 and A_1 and thus $p_j^{A_i} = u_j$, for $i = 0, 1$. When $r_j \le 1$, $\rho_j = u_j$ and thus $p_j^{A_i} = \rho_j$; when $1 < r_j \le x_1$, by Lemma 3 we have $p_j^{A_i} \le x_1\rho_j$. That is, if $r_j \le x_1$, then we always have

$$p_j^{A_i} \le x_1\rho_j, \text{ for } i = 0, 1. \tag{10}$$

If $r_j > y_1$, then J_j is tested in both A_0 and A_1, and thus by Lemma 2 and Eq. (9) we have

$$p_j^{A_i} \le (1 + \frac{1}{y_1})\rho_j = x_1\rho_j, \text{ for } i = 0, 1. \tag{11}$$

If $x_1 < r_j \le \varphi$, then J_j is untested in A_0 but tested in A_1. By Lemmas 2–4, we have

$$p_j^{A_0} \le \varphi\rho_j,\ p_j^{A_1} \le (1 + \frac{1}{x_1})\rho_j, \text{ and } E[p_j^A] \le \max\left\{1 - \alpha + \alpha\varphi, 1 + \frac{1 - \alpha}{x_1}\right\}\rho_j. \tag{12}$$

Lastly, if $\varphi < r_j \le y_1$, then J_j is tested in A_0 but untested in A_1. By Lemmas 2–4, we have

$$p_j^{A_0} \le \varphi\rho_j,\ p_j^{A_1} \le y_1\rho_j, \text{ and } E[p_j^A] \le \max\left\{\alpha + (1 - \alpha)y_1, 1 + \frac{\alpha}{\varphi}\right\}\rho_j. \tag{13}$$

The processing time of J_j in the two algorithms in all the four cases, in Eqs. (10–13), can be merged as

$$p_j^{A_0} \le \varphi\rho_j \text{ and } p_j^{A_1} \le \max\left\{1 + \frac{1}{x_1}, y_1\right\}\rho_j = y_1\rho_j; \tag{14}$$

and the expected processing time of J_j in the revised GCL algorithm in all the four cases can be merged as

$$E[p_j^A] \le \max\left\{x_1, 1 + \frac{1 - \alpha}{x_1}, \alpha + (1 - \alpha)y_1\right\}\rho_j = x_1\rho_j. \tag{15}$$

Suppose the job J_{n_i} determines the makespan of the schedule produced by the algorithm A_i, for $i = 0, 1$, respectively. Note that J_{n_i} is assigned to the least loaded machine by the algorithm. Therefore,

$$C^{A_i} \le \frac{1}{2}\sum_{j=1}^{n_i - 1} p_j^{A_i} + p_{n_i}^{A_i} = \frac{1}{2}\sum_{j=1}^{n_i} p_j^{A_i} + \frac{1}{2}p_{n_i}^{A_i}. \tag{16}$$

We distinguish the following three cases.

Case 1: $n_1 = n_2$. Using Eqs. (14, 15, 16) and the lower bounds in Lemma 5, we have

$$\begin{aligned} E[C^A] &= \alpha C^{A_0} + (1-\alpha) C^{A_1} \\ &\le \frac{1}{2}\sum_{j=1}^{n_1}(\alpha p_j^{A_0} + (1-\alpha) p_j^{A_1}) + \frac{1}{2}(\alpha p_{n_1}^{A_0} + (1-\alpha) p_{n_1}^{A_1}) \\ &= \frac{1}{2}\sum_{j=1}^{n_1} E[p_j^A] + \frac{1}{2} E[p_{n_1}^A] \\ &\le \frac{x_1}{2}\sum_{j=1}^{n_1}\rho_j + \frac{x_1}{2}\rho_{n_1} \\ &\le \frac{3x_1}{2} C^*. \end{aligned}$$

Case 2: $n_1 > n_2$. Eq. (16 becomes

$$C^{A_0} \le \frac{1}{2}\sum_{j=1}^{n_1} p_j^{A_0} + \frac{1}{2} p_{n_1}^{A_0}, \text{ and } C^{A_1} \le \frac{1}{2}\sum_{j=1}^{n_1} p_j^{A_1} + \frac{1}{2} p_{n_2}^{A_1} - \frac{1}{2} p_{n_1}^{A_1}.$$

Then using $p_{n_1}^{A_1} \ge \rho_{n_1}$, Eqs. (9, 14, 15) and the lower bounds in Lemma 5, we have

$$\begin{aligned} E[C^A] &= \alpha C^{A_0} + (1-\alpha) C^{A_1} \\ &\le \frac{1}{2}\sum_{j=1}^{n_1} E[p_j^A] + \frac{\alpha}{2} p_{n_1}^{A_0} + \frac{1-\alpha}{2} p_{n_2}^{A_1} - \frac{1-\alpha}{2} p_{n_1}^{A_1} \\ &\le \frac{x_1}{2}\sum_{j=1}^{n_1}\rho_j + \frac{\alpha\varphi}{2}\rho_{n_1} + \frac{1-\alpha}{2} y_1 \rho_{n_2} - \frac{1-\alpha}{2}\rho_{n_1} \\ &= \frac{x_1}{2}\sum_{j=1}^{n_1}\rho_j + \frac{\varphi-1}{2}\rho_{n_1} + \frac{x_1-\varphi+1}{2}\rho_{n_2} \\ &\le \frac{3x_1}{2} C^*. \end{aligned}$$

Case 3: $n_1 < n_2$. Eq. (16 becomes

$$C^{A_0} \le \frac{1}{2}\sum_{j=1}^{n_2} p_j^{A_0} + \frac{1}{2} p_{n_1}^{A_0} - \frac{1}{2} p_{n_2}^{A_0}, \text{ and } C^{A_1} \le \frac{1}{2}\sum_{j=1}^{n_2} p_j^{A_1} + \frac{1}{2} p_{n_2}^{A_1}.$$

Then using $p_{n_2}^{A_0} \ge \rho_{n_2}$, Eqs. (9, 14, 15) and the lower bounds in Lemma 5, we have

$$\begin{aligned} E[C^A] &= \alpha C^{A_0} + (1-\alpha) C^{A_1} \\ &\le \frac{1}{2}\sum_{j=1}^{n_2} E[p_j^A] + \frac{\alpha}{2} p_{n_1}^{A_0} + \frac{1-\alpha}{2} p_{n_2}^{A_1} - \frac{\alpha}{2} p_{n_2}^{A_0} \\ &\le \frac{x_1}{2}\sum_{j=1}^{n_2}\rho_j + \frac{\alpha\varphi}{2}\rho_{n_1} + \frac{1-\alpha}{2} y_1 \rho_{n_2} - \frac{\alpha}{2}\rho_{n_2} \\ &= \frac{x_1}{2}\sum_{j=1}^{n_2}\rho_j + \frac{1}{2}\rho_{n_1} + \frac{x_1-2\varphi+2}{2}\rho_{n_2} \\ &\le \left(\frac{3x_1}{2} + \frac{3-2\varphi}{2}\right) C^* \\ &\le \frac{3x_1}{2} C^*. \end{aligned}$$

In all the above three cases, we have $E[C^A] \leq \frac{3x_1}{2} C^* = \frac{3\varphi+3\sqrt{13-7\varphi}}{4} C^*$. This proves that the expected competitive ratio of the revised GCL algorithm is at most $\frac{3\varphi+3\sqrt{13-7\varphi}}{4} \approx 2.1839$.

We next give a three-job instance of the two-machine problem $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$ in Instance 4 to show the expected competitive ratio of the revised GCL algorithm is tight.

Instance 4 *In this three-job instance of* $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$*, the job order is* $\langle J_1, J_2, J_3 \rangle$*, with*

$$\begin{cases} u_1 = x_1, & t_1 = 1, \quad p_1 = 0, \\ u_2 = x_1, & t_2 = 1, \quad p_2 = 0, \\ u_3 = 2x_1, & t_3 = 2, \quad p_3 = 0. \end{cases}$$

One sees that for this instance, $\rho_1 = \rho_2 = 1$ and $\rho_3 = 2$. Therefore, the optimal offline makespan is $C^* = 2$ by testing all the jobs and assigning J_1, J_2 on one machine and J_3 on the other machine. On the other hand, $r_j = x_1 < \varphi$ for any $j = 1, 2, 3$ and so they are all untested in either algorithm of A_0 and A_1. It follows that $C^{A_0} = C^{A_1} = 3x_1$ and thus $E[C^A] = 3x_1$ too. The expected competitive ratio of GCL on this instance is $\frac{3x_1}{2}$. This finishes the proof of the theorem. □

We remark that our GCL algorithm for $P \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$ has its expected competitive ratio beating the best known deterministic competitive ratio of $\varphi(2-\frac{1}{m})$, but far away from the proven lower bound of 2 on the deterministic competitive ratio [3]. It is further away from the proven lower bounds of 1.6522 in Theorem 3, 1.6682 in Theorem 2, and $2 - \frac{1}{m}$ [3] on the expected competitive ratio at the presence of two, three and $m \geq 4$ machines, respectively.

Recall that besides the lower bound of 2 on the deterministic competitive ratio for the uniform testing case $P \mid online, t_j = 1, 0 \leq p_j \leq u_j \mid C_{\max}$, Albers and Eckl [3] also showed a slightly better lower bound of 2.0953 for the two-machine general testing case $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$. We improve the latter lower bound of 2.0953 to 2.2117 in the next theorem. Therefore, for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, the expected competitive ratio of GCL not only beats the best known deterministic competitive ratio of 2.3019 [11] for the semi-online variant $P2 \mid t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, but also beats the lower bound of 2.2117 on the deterministic competitive ratio for the fully online problem.

Theorem 6 *The competitive ratio of any deterministic algorithm for* $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{max}$ *is greater than* 2.2117*.*

Proof. Note that we are proving a lower bound on the deterministic competitive ratio, and we use an adversarial argument to construct a three-job instance with the job order $\langle J_1, J_2, J_3 \rangle$. Consider a deterministic algorithm A.

The first job J_1 comes with $u_1 = \varphi$ and $t_1 = 1$. If A tests J_1, then the adversary sets $p_1 = \varphi$. It follows that $p_1^A = \varphi + 1$ and $\rho_1 = \varphi$. If J_1 is untested, then the adversary sets $p_1 = 0$ and thus $p_1^A = \varphi$ and $\rho_1 = 1$. Either way, we have $\frac{p_1^A}{\rho_1} = \varphi$, and thus we assume below that $p_1^A = \varphi$ and $\rho_1 = 1$. (If $p_1^A = \varphi + 1$ and $\rho_1 = \varphi$, then the below u_j- and t_j-values associated with J_2 and J_3 are scaled up by multiplying a factor of φ.)

For any upcoming job J_j, i.e., $j = 2, 3$, the adversary always sets $p_j = u_j$ if J_j is tested by the algorithm, or otherwise sets $p_j = 0$.

Let $x_0 = \frac{3\varphi+1-\sqrt{11\varphi+6}}{2} \approx 0.4878$, which is a root to the quadratic equation $x^2 - (3\varphi+1)x + \varphi^2 = 0$. The second job J_2 comes with $u_2 = \varphi - x_0$ and $t_2 = x_0$. We distinguish two cases on whether or not J_2 is tested by the algorithm.

Case 1: J_2 is tested by the algorithm. In this case, $p_2 = u_2$ and thus $p_2^A = \varphi$ and $\rho_2 = \varphi - x_0$. If J_2 is scheduled on the same machine with J_1, then the third job J_3 is voided (by setting $u_j = t_j = 0$) leading to $C^A = p_1^A + p_2^A = 2\varphi$. Note that $\rho_2 > 1$ and thus $C^* = \rho_2 = \varphi - x_0$. It follows that the competitive ratio is $\frac{2\varphi}{\varphi - x_0}$.

Below we assume J_j is assigned to the machine M_j, for $j = 1, 2$, respectively, and thus the load of each machine is φ. The third job J_3 comes with $u_3 = y_0$ and $t_3 = \varphi + 1 - x_0$, where $y_0 = \frac{1-x_0+\sqrt{(3\varphi-5)x_0+15\varphi+8}}{2} \approx 3.0933$ is a root to the quadratic equation $y^2 - (1-x_0)y - (\varphi+1-x_0)(2\varphi+1-x_0) = 0$.

If J_3 is untested, then $p_3 = 0$ and thus $p_3^A = y_0$ and $\rho_3 = \varphi + 1 - x_0$, leading to $C^A = \varphi + y_0$. In the optimal offline schedule, J_1 and J_2 are scheduled on one machine while J_3 is scheduled on the other, leading to the optimal offline makespan $C^* = \varphi + 1 - x_0$. Therefore, $\frac{C^A}{C^*} = \frac{y_0+\varphi}{\varphi+1-x_0}$.

If J_3 is tested, then $p_3 = u_3$ and thus $p_3^A = t_3 + u_3 = \varphi + 1 - x_0 + y_0$ and $\rho_3 = y_0$, leading to $C^A = 2\varphi + 1 - x_0 + y_0$. In the optimal offline schedule, J_1 and J_2 are scheduled on one machine while J_3 is scheduled on the other, leading to $C^* = y_0$ and subsequently $\frac{C^A}{C^*} = \frac{2\varphi+1-x_0+y_0}{y_0}$.

To conclude this case, we have

$$\frac{C^A}{C^*} \geq \min\left\{\frac{2\varphi}{\varphi - x_0}, \frac{y_0+\varphi}{\varphi+1-x_0}, \frac{2\varphi+1-x_0+y_0}{y_0}\right\}.$$

Since y_0 is a root to the quadratic equation $y^2 - (1-x_0)y - (\varphi+1-x_0)(2\varphi+1-x_0) = 0$, we have

$$y_0(y_0+\varphi) = (\varphi+1-x_0)(y_0+2\varphi+1-x_0),$$

that is, the last two quantities in the above are equal and approximately 2.21172. The first quantity in the above is about 2.8634. Therefore, $\frac{C^A}{C^*} > 2.2117$.

Case 2: J_2 is untested by the algorithm. In this case, $p_2 = 0$ and thus $p_2^A = \varphi - x_0$ and $\rho_2 = x_0$. If J_2 is scheduled on the same machine with J_1, then the third job J_3 is voided (by setting $u_j = t_j = 0$) leading to $C^A = p_1^A + p_2^A = 2\varphi - x_0$. Note that $\rho_2 = x_0 < 1$ and thus $C^* = 1$. It follows that the competitive ratio is $\frac{C^A}{C^*} = 2\varphi - x_0$.

Below we assume J_j is assigned to the machine M_j, for $j = 1, 2$, respectively, and thus the loads of the two machines are φ and $\varphi - x_0$, respectively. The third job J_3 comes with $u_3 = \frac{(1+\varphi)y_0}{2\varphi+1-x_0} \approx 2.1606$ and $t_3 = 1 + x_0$, where y_0 is set the same as in Case 1.

If J_3 is untested, then $p_3 = 0$ and thus $p_3^A = u_3 = \frac{(1+\varphi)y_0}{2\varphi+1-x_0}$ and $\rho_3 = 1 + x_0$, leading to $C^A \geq \varphi - x_0 + \frac{(1+\varphi)y_0}{2\varphi+1-x_0}$ no matter which machine J_3 is assigned to. In the optimal offline schedule, J_1 and J_2 are scheduled on one machine while J_3 is scheduled on the other, leading to the optimal offline makespan $C^* = 1 + x_0$. Therefore, $\frac{C^A}{C^*} \geq \frac{\varphi - x_0 + \frac{(1+\varphi)y_0}{2\varphi+1-x_0}}{1+x_0} = \frac{(\varphi-x_0)(2\varphi+1-x_0)+(1+\varphi)y_0}{(1+x_0)(2\varphi+1-x_0)}$. Since x_0 is a root to the quadratic equation $x^2 - (3\varphi+1)x + \varphi^2 = 0$, we have

$$(\varphi - x_0)(2\varphi + 1 - x_0) = x_0^2 - (3\varphi+1)x_0 + \varphi(1+2\varphi) = \varphi(1+\varphi)$$

and

$$(1 + x_0)(2\varphi + 1 - x_0) = -x_0^2 + 2\varphi x_0 + 2\varphi + 1 = (\varphi+1)(1+\varphi-x_0).$$

It follows that $\frac{C^A}{C^*} \geq \frac{y_0+\varphi}{1+\varphi-x_0}$.

If J_3 is tested, then $p_3 = u_3$ and thus $p_3^A = t_3 + u_3 = 1 + x_0 + \frac{(1+\varphi)y_0}{2\varphi+1-x_0}$ and $\rho_3 = \frac{(1+\varphi)y_0}{2\varphi+1-x_0}$, leading to $C^A \geq 1 + \varphi + \frac{(1+\varphi)y_0}{2\varphi+1-x_0}$ no matter which machine J_3 is assigned to. Since $\rho_3 > \rho_1 + \rho_2$, in the optimal offline schedule, J_1 and J_2 are scheduled on one machine while J_3 is scheduled on the other, leading to $C^* = \rho_3 = \frac{(1+\varphi)y_0}{2\varphi+1-x_0}$ and subsequently $\frac{C^A}{C^*} \geq \frac{y_0+2\varphi+1-x_0}{y_0}$.

To conclude this case, we have

$$\frac{C^A}{C^*} \geq \min\left\{2\varphi - x_0, \frac{y_0+\varphi}{1+\varphi-x_0}, \frac{2\varphi+1-x_0+y_0}{y_0}\right\}.$$

The same as in Case 1, the last two quantities in the above are equal and approximately 2.21172. The first quantity in the above is about 2.7482. Therefore, $\frac{C^A}{C^*} > 2.2117$.

The above two cases prove that the competitive ratio of any deterministic algorithm for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{max}$ is greater than 2.2117. □

5 Conclusion

We investigated the fully online multiprocessor scheduling with testing problem $P \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, and presented a randomized algorithm GCL that is a non-uniform distribution of arbitrarily many deterministic algorithms. To the best of our knowledge, randomized algorithms as meta algorithms in the literature are mostly uniform distributions of their component deterministic algorithms, while our GCL is one to bias towards one component algorithm; also, when there are many component algorithms, the previous randomized algorithm often involves bookkeeping all their solutions, while our GCL does not do so since the component algorithms are independent of each other. We showed that the expected competitive ratio of our GCL is around 3.1490. When there are only two machines, i.e., for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, using only two component algorithms in the revised GCL algorithm leads to an expected competitive ratio of 2.1839.

We also proved three inapproximability results, including a lower bound of 1.6682 on the expected competitive ratio of any randomized algorithm at the presence of at least three machines, a lower bound of 1.6522 on the expected competitive ratio of any randomized algorithm for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, and a lower bound of 2.2117 on the competitive ratio of any deterministic algorithm for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$. By the last lower bound, we conclude that the algorithm GCL beats any deterministic algorithm for $P2 \mid online, t_j, 0 \leq p_j \leq u_j \mid C_{\max}$, in terms of its expected competitive ratio. Such an algorithmic result is rarely seen in the literature.

The expected competitive ratio of the algorithm GCL is far away from the lower bounds of 1.6522, 1.6682 and $2 - \frac{1}{m}$ for two, three and $m \geq 4$ machines, respectively, suggesting future research to narrow the gaps, even for the two-machine case. One sees that the testing probability function in the job ratio in the algorithm GCL is non-continuous at φ, suggesting the existence of a plausible better probability distribution function for choosing component algorithms.

Declarations

Data availability. Not applicable.

Interests. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Albers. On randomized online scheduling. In *Proceedings of ACM STOC 2002*, pages 134–143, 2002.

[2] S. Albers and A. Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In *Proceedings of WAOA 2020*, LNCS 12806, pages 127–142, 2020.

[3] S. Albers and A. Eckl. Scheduling with testing on multiple identical parallel machines. In *Proceedings of WADS 2021*, LNCS 12808, pages 29–42, 2021.

[4] Y. Bartal, M. Chrobak, and L. L. Larmore. A randomized algorithm for two servers on the line. *Information and Computation*, 158:53–69, 2000.

[5] Y. Bartal, A. Fiat, H. J. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and Systems Science*, 51:359–366, 1995.

[6] C. Dürr, T. Erlebach, N. Megow, and J. Meißner. Scheduling with explorable uncertainty. In *Proceedings of ITCS 2018*, LIPIcs 94, pages 30:1–30:14, 2018.

[7] C. Dürr, T. Erlebach, N. Megow, and J. Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82:3630–3675, 2020.

[8] R. Fleischer and M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.

[9] S. Fung, C. Poon, and F. Zheng. Online interval scheduling: Randomized and multiprocessor cases. *Journal of Combinatorial Optimization*, 16:248–262, 2008.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

[11] M. Gong, J. Fan, G. Lin, B. Su, Z. Su, and X. Zhang. Multiprocessor scheduling with testing: Improved approximation algorithms and numerical experiments. *arXiv*, 2204.03299, 2022.

[12] M. Gong and G. Lin. Improved approximation algorithms for multiprocessor scheduling with testing. In *Proceedings of FAW 2021*, LNCS 12874, pages 65–77, 2021.

[13] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, 45:1563–1581, 1966.

[14] N. Reingold, J. R. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.

[15] S. S. Seiden. Online randomized multiprocessor scheduling. *Algorithmica*, 28:173–216, 2000.

[16] S. S. Seiden. Barely random algorithms for multiprocessor scheduling. *Journal of Scheduling*, 6:309–334, 2003.

[17] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of IEEE FOCS 1977*, pages 222–227, 1977.

毕业论文（设计）文献综述和开题报告考核

对文献综述、外文翻译和开题报告评语及成绩评定：

成绩比例	文献综述 占（10%）	开题报告 占（15%）	外文翻译 占（5%）
分值			

开题报告答辩小组负责人（签名）_____

年　月　日