arXiv:2603.28800v1 [cs.GT] 26 Mar 2026

时间依赖效用调度：公平性与效率

Gaia Nicosia* Andrea Pacifici† Ulrich Pferschy‡

2026年4月1日

摘要

本文介绍一类新的多智能体单机调度问题，其中每个作业都关联一个自利智能体，其效用函数随完工时间增加而递减。我们的目标是最大化所有代理中的最小效用，从而实现公平解。我们研究了该问题的复杂性，并针对若干变体提出求解方法。对于一般情形，我们提出一种二分搜索过程，用以找到可能达到的最大最小效用，并给出一种精确的基于贪心的替代方法。我们分析了带释放日期和交期的变体，结果表明：任意释放日期的情形具有强 NP 困难性；只有一个作业具有释放日期时则为弱 NP 困难；当所有作业的处理时间相同时，问题可在多项式时间内求解。对于上述所有情形，我们还研究了相应的高效解问题，即最大化效用总和。

我们还考察了可在预算约束内调整线性效用函数的情形，探讨修改截距或斜率对最优调度的影响。从单智能体视角出发，我们研究了提升某个代理的效用对整体解的影响。若要插入一个效用尽可能高的新作业，则会产生重新调度问题；这类问题需施加不同的下界，具体取决于原公平调度的效用。最后，我们考虑一种双层设置：领导者通过修改效用函数来强制实现某个目标调度，而跟随者则针对修改后的实例计算公平解。我们的研究推动了调度理论、多智能体系统和算法公平性的发展，并凸显了竞争性调度中以公平为导向的目标。

关键词：调度；多代理；公平性；复杂性理论

1 引言

调度问题长期以来一直是《运筹学》和计算机科学中的核心课题，应用范围涵盖制造业、物流、云计算和服务运营等领域。经典模型通常假设由单一决策者优化整个系统的

*土木工程、计算机科学与航空技术系，罗马第三大学，意大利罗马，Vasca Navale 路 79 号，邮编 00146，电子邮件：gaia.nicosia@uniroma3.it

†土木工程与计算机工程系，罗马第二大学“Tor Vergata”，意大利罗马，德尔波利特尼科路 1 号，邮编 00133，电子邮件：andrea.pacifici@uniroma2.it

‡格拉茨大学运营与信息系统系，奥地利格拉茨 Universitaetsstrasse 15 号，邮编 8010。电子邮件：ulrich.pferschy@uni-graz.at

性能。近几十年来，多代理调度问题日益受到关注，其中多个自主智能体竞争共享资源的使用权——最常见的是一台单机。

多代理调度研究始于 Baker 和 Smith（2003年）以及 Agnetis 等人（2004年）的开创性工作，这些工作为涉及多个可能具有冲突目标的决策者的调度问题建模奠定了基础。此后，大量文献研究了这样的情形：代理控制多个作业，并试图最小化各自基于完工时间的成本函数（例如，参见 Agnetis 等人，2025年、2019年、2015年；Hermelin 等人，2025年；Wellman 等人，2001年）。

近来，研究者越来越关注一种特殊情形：每个作业对应一个独立的代理（Niu 等人，2023年）。这种表述刻画了偏好高度个性化的场景，主要通过经典非合作博弈论模型加以研究，其中代理选择诸如并行机环境中的机器选择等策略（例如，Correa 和 Queyranne，2012年；Hoeksma 和 Uetz，2019年）。然而，仅考虑策略行为并未顾及集体福利，且可能导致结果严重失衡，使得部分代理承担高得多的成本。

在本研究中，我们考察一类公平性驱动的多智能体单机调度问题，其中每项作业对应一个自利智能体，其效用函数取决于该作业的完工时间。我们不通过功利主义或纳什型目标来汇总效用，而是关注最大最小公平准则，即最大化所有代理中的最小效用（或等价地，最小化最大成本）。这一目标确保没有代理受到过度惩罚，从而促进平衡且为社会所接受的调度方案；在公平与公平性发挥核心作用的竞争环境中，这一点尤为重要。

在许多共享资源调度场景中，代理会因其活动完成而获得时间敏感型效用，延迟则会导致价值单调损失（Raut et al., 2008a,b）。这一现象自然出现在经济环境中：收益会随时间折现，因此较晚完工会因机会成本、不确定性或资本未能用于其他用途而导致净现值降低。当共同资源的使用受到限制时，仅关注总体绩效的目标可能造成完工时间极不均衡，使某些代理受到过度惩罚。因此，最大最小效用准则提供了一种有原则的机制，通过明确限制调度决策对任何个体参与者造成的不利影响程度，来缓解此类失衡。

在需求或支付意愿迅速衰减的市场情境中，也存在类似的考量。对于商业生命周期较短或初期炒作热度很高的产品，延迟生产或交付所带来的收入可能会随时间大幅下降（Dedehayir 和 Steinert，2016年）。在这种情况下，最大化总效用仍可能导致某些代理遭遇延误，使其参与在经济上变得无足轻重。相较之下，优化最小效用能够促进更公平的完工时间分配，确保所有代理都能保有可实现价值中不可忽视的一部分，同时兼顾对时间相关经济效应的敏感性。

我们研究了该问题的若干变体，考虑不同的代理特定效用函数和时间约束，例如释放日期和交期。我们还分析

了问题的若干版本，其中某些输入参数可以进行调整，以获得特定类别的解，或分析这些修改如何影响最终解。我们的主要贡献如下：

- 我们提出了精确的求解方法，包括一种用于确定可实现的最大最小效用的二分搜索框架，以及一种适用范围广泛的贪心替代方案。
- 我们阐明了这些变体的复杂性结果。例如，当存在任意释放日期时，问题具有强 NP 困难性；当仅有一个作业受到释放约束时，问题具有弱 NP 困难性。同时，我们也找到了多项式时间可解的情形，例如处理时间相等的情况。

- 我们将该框架扩展到线性效用函数可在预算约束内调整的情形，并分析了对截距或斜率的修改如何影响均衡调度。

- 从单智能体视角出发，我们研究了提高某个代理的效用会如何影响整体的公平导向解，并揭示了个体激励与集体保障之间权衡的结构性特点。

- 我们还研究了领导者-跟随者设定下的问题：领导者通过调整跟随者代理的效用函数来促成目标解；反过来，这些跟随者代理会按照公平调度来安排各自的作业。

本文讨论的各个问题将在第 3 节介绍，该节还将说明本文的组织结构。

2 相关文献

本节回顾与本研究重点密切相关的现有调度问题文献，旨在将我们的工作置于更广泛的研究背景中。首先，我们探讨多代理调度，考察帕累托最优解及与公平性相关的考量因素——这些因素在实际应用中仍居于核心地位；随后转向将作业本身视为代理的情形，最后讨论作业价值随时间变化的调度问题，着重说明动态优先级带来的挑战，以及其为自适应方法创造的机遇。

多代理调度涉及多个代理，每个代理都有各自不同的目标和作业集，并共同竞争共享的处理资源。这种范式广泛应用于各种领域，包括制造系统、云计算和协作机器人；在这些场景中，在相互竞争的实体之间高效分配资源至关重要。

早期研究主要关注涉及两个代理的情形（例如，Agnetis 等人，2004年；Baker 和 Smith，2003年）。然而，现实应用往往涉及两个以上代理的环境，因此需要更复杂的调度策略，以应对异质目标之间更高维度的相互作用。这推动了双代理情形之外多代理调度研究文献的大幅扩展。例如，Agnetis 等人（2007年）分析了多代理单机调度问题，其中每个代理都拥有一组不可抢占作业，并阐述了若干经典性能指标下的关键复杂性结果。同样，Yuan（2017年）研究了代理数量固定的单机模型，

并识别出仍可在多项式时间内求解的问题变体，同时提出了构造性算法解法。同样，Li 和 Yuan（2020年）研究了以总加权延迟工作目标为准则的竞争性多代理调度，并提出了用于确定帕累托最优解的方法，以反映代理之间的权衡。最近，Wang 等人（2021年）研究了云制造平台上的多任务调度模型，该模型涉及多个竞争智能体，并给出了最新的复杂性结果及相关求解程序；Chen 等人（2026年）则研究了若干考虑不同智能体目标、针对等长度作业的单机可行性问题。

在这一背景下，公平性已日益成为一个重要主题，因为纯粹以效率为导向的解可能会使某些代理获得不成比例的利益（Agnetis 等人，2019年）。因此，在多个决策者竞争共享资源的系统中，将公平性纳入调度算法已变得至关重要。近年来，越来越多的文献从不同视角探讨公平性（近期研究成果见 Agnetis 等人，2025年；Hermelin 等人，2025年）。具体而言，Agnetis 等人（2025年）研究了引入公平性约束如何重塑多智能体单机调度问题的结构，并影响效率与公平之间的权衡。与之互补的是，Hermelin 等人（2025年）研究了公平性调度模型，指出公平要求可能带来新的算法挑战，并显著改变可行解集合。

社会福利解（即使系统整体效用最大化的解）在个体效用方面可能极不平衡，因此会令某些代理不满意。另一方面，公平调度的全局效用可能远低于其最大值。由此自然产生一个问题：按照公平调度安排作业时，社会福利的最大损失是多少？对此，相关文献中有相当一部分致力于为所谓的公平代价（PoF）寻找上界。公平代价由 Bertsimas 等人（2011年）提出，随后被多篇论文探讨（Agnetis 等人，2019年；Nicosia 等人，2017年；Zhang 等人，2020年）；在一般资源分配问题的背景下，它表示公平解相对于总效用最大化解的效用损失。近期一项研究针对时变效用模型，在无嫉妒条件下给出了公平代价的紧确界，并揭示了结构参数如何影响公平性导致的效用损失（Vardi 和 Haskell，2025年）。

在相关研究领域中，作业本身可能被视为代理。例如，在合作式调度博弈中，参与者可以交换效用，以稳定全局最优调度（Curiel 等人，1989年）；而在非合作环境中，研究重点则转向激励与公平性约束之间的相互作用，Vardi 和 Haskell（2025年）的研究便对此进行了阐释。

最后一类研究聚焦于作业价值随时间变化的调度问题。Raut 等人（2008a）研究了作业价值递减的情形，分析了一台机器上的问题，其中每项作业的价值会随着其开始时间推迟而降低。他们考察了无约束、截断和容量约束退化函数，并表明，尽管某些特殊情形——例如线性无约束退化——是多项式时间可解的，但一般问题是一元 NP 困难。他们采用时间索引公式对问题建模，并提出了若干启发式算法，对其经验表现进行了细致评估。在这条研究脉络的基础上，Raut 等人（2008b）将模型扩展到容量受限情形，其中

机器同时能够处理的作业数量有限。他们证明，新增的容量约束并未改变问题的 NP 困难性质，并提出了基于乘法分段度量的新启发式算法，其表现优于现有方法。这些研究表明，时变作业效用会带来显著的结构复杂性，因此需要量身定制的算法策略。

3 问题描述

给定一组 n 不可抢占作业 J，我们考虑一类单机调度问题，其中每项作业都与一个代理相关联。对于任意 $j \in J$，给定处理时间p_j、单值非递增的效用函数$u_j(\cdot)$，以及可能存在的释放日期r_j 和交期 d_j。作业 J 的任意调度 σ 都会为每项作业/代理 j确定一个完工时间 $C_j(\sigma)$。我们将调度 σ 中代理 j 的效用定义为$u_j(C_j(\sigma))$。根据上下文，我们也会等价地写作$u_j(C_j)$或$u_j(\sigma)$，或者简单地写作u_j。

不失一般性，我们假设对于所有可行调度，$u_j \geq 0$。通过给所有效用函数加上一个足够大的常数即可轻松实现这一点，即 $M := \max\{0, -u_j(C)\}$，其中 C 是所有完工时间的某个上界，例如 $P := \sum_j p_j$；若存在释放日期r_j，则也可以取$\max_j r_j + \sum_j p_j$。我们有时还会使用以下广义反效用函数（Kampke 和 Radermacher，2015年），它给出达到效用目标值 u^T所允许的最晚完工时间：

$$u_j^{-1}(u^T) := \sup\{C \in \mathbb{R} \mid u_j(C) \geq u^T\}. \tag{1}$$

下文将讨论一般的非递增线性效用函数情形，但有时也将研究范围限定为满足 $u_j(C_j) = b_j - a_j C_j$ with $a_j \geq 0$ 的线性效用函数。

在本文剩余部分中，我们用 $\mathcal{S}$ 表示当前问题的可行半主动调度集合（其中，在不违反给定顺序的情况下，任何作业都不能更早开始）。表1列出了本文剩余部分所讨论的所有问题所采用的符号。

本文所考虑的公平调度问题旨在确定一个公平调度σ_F，即使所有代理中的最小效用最大化的调度方案。令 $u_{\min}(\sigma) := \min_{j \in J} u_j(\sigma)$，则公平调度定义为

$$\sigma_F \in \arg\max_{\sigma \in \mathcal{S}} \{u_{\min}(\sigma)\}, \qquad u_F := u_{\min}(\sigma_F). \tag{2}$$

作为一个重要的特殊情形，当所有代理共享同一个效用函数时，计算 σ_F 就归结为使所有作业的最大完工时间最小化，即经典的最大完工时间目标 $C_{\max}$。

对于给定的问题，除了公平解之外，我们还考虑系统最优解，即使系统的全局效用（或社会福利）最大化的调度 σ_{SYS}。设

$u_{\text{sum}}(\sigma) := \sum_{j\in J} u_j(\sigma)$; 则系统最优调度定义为

$$\sigma_{SYS} \in \arg\max_{\sigma\in\mathcal{S}} \{u_{\text{sum}}(\sigma)\}, \qquad u_{SYS} := u_{\text{sum}}(\sigma_{SYS}). \tag{3}$$

根据代理效用函数的特征，寻找系统最优

$J := \{1,\ldots,n\}$	以 j 为索引的作业/代理集合
p_j, r_j, d_j, w_j	作业 j 的处理时间、释放日期、交期
$P := \sum_j p_j$	作业的总加工时间
$s_j(\sigma)$, s_j	调度 σ 中作业 j 的开始时间
$C_j(\sigma)$, C_j	调度 σ 中作业 j 的完工时间
$u_j(C_j(\sigma))$, $u_j(C_j)$, $u_j(\sigma)$	调度方案 σ 中作业 j 在 C_j 完成时的效用
$u_j(C_j) = b_j - a_j C_j$	作业 j 在 C_j完成时的线性效用
$-a_j$	线性函数的斜率
b_j	线性函数的截距
$\mathcal{S}$	可行（半主动）调度方案集合
$u_{\min}(\sigma) = \text{分钟}_{j\in J}\{u_j(\sigma)\}$	调度 σ 达到的最小效用
$u_{\text{总和}}(\sigma) = \sum_{j\in J} u_j(\sigma)$	调度 σ 的总效用
$\sigma_F = \text{最大化参数}_{\sigma\in\mathcal{S}}\{u_{\min}(\sigma)\}$	公平（最大–最小）解
$\sigma_{SYS} = \arg\max_{\sigma\in\mathcal{S}}\{u_{\text{总和}}(\sigma)\}$	系统最优解
$u_F = u_{\min}(\sigma_F)$	公平（最大最小）解值
$u_{SYS} = u_{\text{求和}}(\sigma_{SYS})$	系统最优解值
Δa_j Δb_j	var
δ_j	扰动

表1：符号说明

调度 σ_{SYS} 可能相当于最小化一个正则目标函数，关于该函数，调度文献中已有许多研究成果。特别是，对于线性效用函数，由于

$$\max \sum_{j\in J} u_j(C_j) = \max \sum_{j\in J} (b_j - a_j C_j) = \sum_{j\in J} b_j - \min \sum_{j\in J} a_j C_j, \tag{4}$$

计算 σ_{SYS} 等价于最小化加权完成时间和。这个问题已得到广泛研究。事实上，在不存在附加约束或特殊系统特征的情况下，当$u_j(C_j) = b_j - a_j C_j$时，可根据Smith规则（也称加权最短处理时间规则），按 a_j/p_j的非递增顺序排列作业，从而获得系统最优解。

显然，在效用函数 $u_j(\cdot)$ 单调不增的一般情形下，寻找使效用总和最大化的调度取决于效用函数的具体形式，可能远非易事。例如，假设代理 j 的效用为 $u_j(C_j) = \min\{d, b_j - a_j C_j\}$. 。在这种情况下，只要代理 j 的任务在“交期”（$b_j - d$）$/a_j$之后完成，其效用就会从 d开始下降 j 。这一表述涵盖了经典调度问题 $1||\sum w_j T_j$（最小化总加权逾期时间），该问题众所周知是强 NP 困难的（参见 Lenstra 等人（1977）第 359 页）。即使 $u_j = b_j - a_j f(C_j)$，且 $f(\cdot)$ 是分段线性函数，求系统最优解仍然是强 NP 困难的（参见 H¨ohn 和 Jacobs，2015 年的定理 5）。有趣的是，当所有代理的 a_j 都相同时，二次情形 $u_j = b_j - a_j C_j^2$ 可在多项式时间内求解，因为 SPT 规则可为标准调度问题 $1||\sum C_j^2$ 给出最优调度（Townsend，1978年）。然而，对于任意智能体权重，复杂性状态仍未解决

在情形 $u_j(C_j) = b_j - a_j C_j^2$ 下仍未解决（Chen 等人，1998 年；H¨ohn 和 Jacobs，2015 年）。总而言之，我们可以得出以下结论：

观察 1 对于代理的一般（分别为线性）非递增效用函数，最大化总效用，即求系统最优解，是强 *NP* 困难的（分别可通过 *Smith*规则在多项式时间内求解）。

值得一提的是，公平解的全局效用可能远小于系统最优解的全局效用；与此同时，公平性值 $u_{\min}(\sigma_{SYS}) = \min_j\{u_j(\sigma_{SYS})\}$（即系统最优解中处境较差的代理的效用）也可能任意小于公平解的公平性值 $u_F = \min_j\{u_j(\sigma_F)\}$。通过一个简单的例子即可说明这一点：即使所有作业的处理时间都相同，$u_{\text{sum}}(\sigma_{SYS}) = \sum_{j\in J} u_j(\sigma_{SYS})$ 与$u_{\text{sum}}(\sigma_F) = \sum_{j\in J} u_j(\sigma_F)$ 的比值仍可任意大。考虑具有 $p_1 = p_2 = 1$、且效用函数为线性函数 $u_1(C_1) = 2M+1-MC_1$和 $u_2(C_2) = 2 - C_2$的 $n = 2$ 个作业。若先调度作业 1，则得到 $u_1 = M + 1$ 和 $u_2 = 0$，由此得到全局最优调度 σ_{SYS}。相反，若先调度作业 2，则得到 $u_2 = 1$ 和 $u_1 = 1$，因此该排序构成公平调度 σ_F。鉴于这一负面结果，本研究不再进一步探讨公平代价问题。

本文根据智能体/作业视角，考虑四类不同的问题。这些问题将在第3.1–3.4节中严格定义。按照调度领域的惯例，我们采用三字段表示法来表示所讨论的问题。上述公平调度问题记为 $1|\beta|u_{\min}$，其特殊性质在中间字段 β中说明。系统最优则通过求解问题 $1|\beta|\sum u_j$得到。下方表2概述了这些问题及其相应表示法。

N.	问题描述	符号说明	章节
1	一般效用，目标 $f = u_{\min}$ 或 $f = \sum u_j$	$1\|u_j\|f$	4.1,4.2
2	线性效用，目标 $f \in \{u_{\min}, \sum u_j\}$	$1\|u_j = b_j - a_jC_j\|f$	4.1,4.2
3	释放日期、线性效用、目标函数 $f \in \{u_{\min}, \sum u_j\}$	$1\|r_j, u_j = b_j - a_jC_j\|f$	4.3
4	释放日期、单位加工时间、线性效用、目标函数 $f \in \{u_{\min}, \sum u_j\}$	$1\|p_j = 1, r_j, u_j = b_j - a_jC_j\|f$	4.3
5	释放日期、相同加工时间、线性效用、目标函数 $f \in \{u_{\min}, \sum u_j\}$	$1\|p_j = p, r_j, u_j = b_j - a_jC_j\|f$	4.3
6	交期、线性效用、迟到作业数量有界、目标值 $f \in \{u_{\min}, \sum u_j\}$	$1\|\sum U_j \le k, u_j = b_j - a_jC_j\|f$	4.4
7	截距增加	$1\|u_j = \tilde{b}_j - a_jC_j\|u_{\min}$	5.1
8	斜率减少	$1\|u_j = b_j - \tilde{a}_jC_j\|u_{\min}$	5.2
9	面积不变的效用变化	$1\|u_j = b_j - \tilde{a}_jC_j,\ A_j = A_j\|u_{\min}$	5.3
10	单个代理的效用变化 i，目标 u_i	$1\|\tilde{u}_i\|u_i$	5.4
11	加入新代理后的重新调度，个体扰动有界 δ_j	$1\|resch,\ \delta_j \ge 0\|u_{n+1}$	6
12	加入新代理后的重新调度，总体扰动有界 δ_j	$1\|resch,\ \sum\delta_j \ge 0\|u_{n+1}$	6
13	加入新代理后的重新调度，调度最优性保持	$1\|resch, u_j \ge u_F\|u_{n+1}$	6
14	目标序列 σ_F（领导者-跟随者设定），使 b_j的增加最小化	$1\|\sigma_F = \sigma_T, u_j = \tilde{b}_j - a_jC_j\|\sum\tilde{b}_j$	7
15	目标序列 σ_F（领导者-跟随者设定），使 a_j的减少最小化	$1\|\sigma_F = \sigma_T, u_j = b_j - \tilde{a}_jC_j\|\sum -\tilde{a}_j$	7
16	目标序列 σ_{SYS}（领导者-跟随者设定），最小化 a_j的下降量	$1\|\sigma_{SYS} = \sigma_T, u_j = b_j - \tilde{a}_jC_j\|\sum -\tilde{a}_j$	7

表2：所研究的问题变体及采用的 Graham记号说明。

3.1 固定参数问题

我们首先讨论一组经典离线单机调度问题，其中代理等同于标准作业，每个作业都由给定的固定参数描述。具体而言，我们考虑该问题的若干变体：有无释放日期和交期、处理时间任意/为单位时间/为常数，以及分别以公平解和系统最优解为目标的两种不同目标函数。这些变体的表示法汇总于表2的前六行，相关算法与复杂性结果见第4节。

3.2 可变线性效用函数

第二类问题涉及效用函数为线性的代理，即 $u_j = b_j - a_j C_j$，其中 $a_j \geq 0$， C_j 表示代理 j的完工时间。可以修改这些效用函数的某些参数，以提高最大最小公平性或系统总效用。为确保调整符合实际，修改受到约束。我们用波浪号标示可修改的参数（见表2第7–10行）。下面将形式化说明我们考虑的变化类型。

提高截距 b_j。我们引入一个模型，其中每个截距 b_j 都允许˜增加一个非负量 Δb_j，从而得到新的截距 $b_j = b_j + \Delta b_j$，同时受总预算约束

$$\sum_j \Delta b_j \leq B.$$

修改后的效用函数形式为 $u_j = (b_j + \Delta b_j) - a_j C_j$。形式化地说，优化问题为

$$\max_{\Delta b_j \geq 0} \ \min_j \big((b_j + \Delta b_j) - a_j C_j\big) \quad \text{s.t.} \ \sum_j \Delta b_j \leq B.$$

˜该问题在第5.1节中研究，记作 $1|u_j = b_j - a_j C_j|u_{\min}$。

降低斜率a_j。类似地，我们考虑将斜率 a_j 降低非负量 Δa_j，并满足以下条件：

$$0 \leq \Delta a_j \leq a_j, \quad \sum_j \Delta a_j \leq A,$$

使得具有新斜率 $\tilde{a}_j = a_j - \Delta a_j$ 的修改后效用在 C_j 中仍保持非递增。修改后的效用函数变为 $u_j = b_j - (a_j - \Delta a_j) C_j$。我们在第 5.2 节讨论这个问题，并将其记为 $1|u_j = b_j - \tilde{a}_j C_j|u_{\min}$。

面积约束下的变化。我们考虑对线性效用模型的一种变体，其中截距 b_j 和斜率 a_j 允许同时变化，但需满足每个代理所取期望效用值的聚合约束。

假设作业的完工时间是一个服从均匀分布 $C_j \sim U(0, P)$ 的随机变量，那么，根据线性性质，代理 j 的期望效用值为 $\mathbb{E}[u_j] = b_j - \tilde{a}_j \mathbb{E}[C_j] = \tilde{b}_j - \tilde{a}_j \frac{P}{2}$。我们希望在保持 $\mathbb{E}[u_j]$ 不变的情况下选择 $\tilde{a}_j$ 和 $\tilde{b}_j$，尤其是使其等于原始值 $b_j - a_j \frac{P}{2}$。由于由效用直线

与水平轴围成的梯形面积在 $[0,P]$ 上为

$$A_j \;=\; \int_0^P \big(b_j - a_j t\big)\,dt \;=\; P\big(b_j - a_j\frac{P}{2}\big) \;=:\; P\big(\tilde{b}_j - \tilde{a}_j\frac{P}{2}\big) = \tilde{A}_j,$$

我们的约束等价于要求梯形面积保持不变。

此外，为确保线性效用函数在可行完工时间的整个区间内保持非递增且非负，我们施加 $\tilde{a}_j \geq 0$ and

$$\tilde{b}_j - \tilde{a}_j t \;\geq 0, \quad \forall\, t \in [0,P],$$

˜这等价于 $b_j \geq \tilde{a}_j P$。总之，我们用调度表示法将该问题记为˜ ˜1|$u_j = b_j - \tilde{a}_j C_j$, $A = A_j|u_{\min}$ 并在第 5.3 节中讨论。

单个代理的变体。我们还考虑这样一种情况：只有单个代理，例如代理i，可以调整其效用函数，而所有其他代理的数据保持不变。共有以下三种情形：

1. 截距 b_i 可以增加；2. 斜率 $a_i > 0$ 可以减小（同时保持非负）；
3. 面积 A_i 可以增加。

对于每种变体，我们分析效用 u_i 如何变化，以及最优调度如何调整，从而改善 u_i 或总体目标。我们将这三个问题版本˜ ˜ ˜分别记为 $1|u_i = b_i - a_i C_i|u_{\bar{j}}$、$1|u_i = b_i - \tilde{a}_i C_i|u_{\bar{j}}$ 和 $1|u_i = b_i - \tilde{a}_i C_i,\ A = A_i|u_{\bar{j}}$ 并在第5.4节中研究。

3.3 从新代理的视角重新调度

与第3.2节中介绍的单代理变体类似，我们从新代理的视角考虑第三类问题。

假设初始问题已得到最优解，从而得到一个调度 σ_F，其中每个作业 $j = 1,\dots,n$ 对应的效用值为 $u_j(\sigma_F)$，且最大最小效用为 $u_F := \min_j\{u_j(\sigma_F)\}$。随后，我们考虑一个重新调度阶段，在此阶段中引入一个新代理，记为 $n+1$。目标是确定一个修订后的调度 σ，使得：(1) 新代理 $u_{n+1}(\sigma)$ 的效用最大化；(2) 保持原有代理 $j = 1,\dots,n$的某些公平条件。

形式化地，给定初始调度 σ_F 和原有的效用 $u_j(\sigma_F)$，该问题为

$$\max_{\sigma}\; u_{n+1}(\sigma)$$

满足针对其余代理的公平性保持约束。

为实现公平性，我们允许向每个原有代理提供非负的 补偿 $\beta_j \geq 0$，例如以金钱或资源调整的形式。具体而言，我们考虑对原有代理的效用（包括其补偿）施加三种公平性约束：

1. 个体不减：对于所有 $j = 1$、.....、 n，$u_j(\sigma) + \beta_j \geq u_j(\sigma_F)$；2. 总体不减：$\sum_{j=1}^n \big(u_j(\sigma) + \beta_j\big) \geq \sum_{j=1}^n u_j(\sigma_F)$；

3. 调度最优性保持：新调度中所有 n 原代理的最小效用（包括补偿）至少为 u_F。

这些表述具有一般性：如果从某个原始 n 代理而非新引入的代理的角度进行重新调度，同样的推理依然适用。我们将新调度 σ中任意作业 j 相较于原解 σ_F的效用扰动记为 $\delta_j = u_j(\sigma) + \beta_j - u_j(\sigma_F)$，其中包括补偿。

于是，这个问题的三个所述版本分别记为 $1|resch、\ \delta_j \geq 0|u_{n+1}$,$1|resch、\ \sum \delta_j \geq 0|u_{n+1}$ 和 $1|resch、\ u_j \geq u_F|u_{n+1}$。关于这类问题的计算和分析结果，请参见第6节。

3.4 领导者-跟随者效用调整

最后，我们考虑一种受领导者-跟随者场景启发的双层优化设定。某个实体——领导者——旨在强制实施特定调度 σ_T，例如，使完工时间总和最小化以优化吞吐量，或使总效用最大化。在这种设定下，跟随者仍会求解所得问题 $1|\ldots|u_{\min}$，以便为所有作业找到公平解；但领导者可以通过修改作业的效用函数来影响结果。为此，~领导者可以控制截距 $b_j = b_j + \Delta b_j$或斜率 $\tilde{a}_j = a_j - \Delta a_j$，方式类似于前文所述的修改。

这些变更 $\Delta b_j \geq 0$ 和 $0 \leq \Delta a_j \leq a_j$ 会影响跟随者在计算公平解时确定的序列。因此，我们关注以下问题：最小化总修改成本，即 $\sum_j \Delta b_j$ 或 $\sum_j \Delta a_j$，条件是跟随者（其目标是在修改后的效用函数下最大化最小效用）产生的公平解 σ_F 对应于所需的目标序列 σ_T。

~ ~我们将这两个问题版本分别记为 $1|\sigma^* = \sigma_T,\ u_j = b_j - a_j C_j|\sum b_j$和 $1|\sigma^* = \sigma_T,\ u_j = b_j - \tilde{a}_j C_j|\sum -\tilde{a}_j$其中 σ^*可以是公平解 σ_F或系统最优 σ_{SYS}，并在第7节中对它们进行研究。（如该节所述，注意，问题 $1|\sigma_{SYS} = \sigma_T,\ u_j =_{\sim} \quad b_j - a_j C_j|\sum b_j$没有实际意义。）

3.5 贡献总结

对于本文考察的十六种（若将前六个问题各自的两种不同目标也考虑在内，则为二十二种）问题变体，我们对其计算复杂性进行了详细分析，并在可能的情况下提供了高效的求解方法。具体而言，对于每一种变体，我们要么证明其为 NP难性，要么设计多项式时间算法，有时还设计伪多项式时间算法。

表 3 概述了我们的主要贡献，重点展示了各个问题设定的复杂性分类以及所获得的算法结果类型。本总结旨在引导读者了解本文分析的各种模型以及取得的方法论进展。

请注意，我们在第 4 节和第 6 节中的所有算法都适用于一般的非递增效用函数（假设函数值可以在常数时间内计算）。

问题	针对 $f = u_{\min}$ 的复杂性	针对 $f = \sum u_j$ 的复杂性
$1\|u_j\|f$	多项式时间可解（定理 2）	强 NP 困难（观察 1）
$1\|u_j = b_j - a_jC_j\|f$	多项式时间可解（定理 2）	多项式时间可解（Smith，1956年）
$1\|r_j,\ u_j = b_j - a_jC_j\|f$	强 NP 困难（定理 3）	强 NP 困难（定理 5）
$1\|p_j = 1,\ r_j,\ u_j\|f$	多项式时间可解（定理 7）	多项式时间可解：指派问题（观察 8）
$1\|p_j = p,\ r_j,\ u_j\|f$	多项式时间可解（定理 9）	多项式时间可解（参见 Baptiste，2000，观察 10）
$1\|\sum U_j \le k,\ u_j = b_j - a_jC_j\|f$	多项式时间可解（定理 12）	强 NP 困难（定理 13）

表 3：表2中问题 1–6 的结果概述

4 固定参数问题

问题	复杂性
$1\|u_j = \tilde{b}_j - a_jC_j\|u_{\min}$	多项式时间可解（第 5.1 节）
$1\|u_j = b_j - \tilde{a}_jC_j\|u_{\min}$	多项式时间可解（第 5.2 节）
$1\|u_j = b_j - \tilde{a}_jC_j,\ A_j = A_j\|u_{\min}$	多项式时间可解（第5.3节）
$1\|\tilde{u}_i\|u_i$	多项式时间可解（第5.4节）
$1\|resch,\ \delta_j \ge 0\|u_{n+1}$	弱 NP 困难（定理14）
$1\|resch,\ \sum\delta_j \ge 0\|u_{n+1}$	弱 NP 困难（定理15）
$1\|resch,\ u_j \ge u_F\|u_{n+1}$	多项式时间可解（定理16）
$1\|\sigma_F = \sigma_T,\ u_j = b_j - a_jC_j\|\sum b_j$	多项式时间可解（第7节）
$1\|\sigma_F = \sigma_T, u_j = b_j - \tilde{a}_jC_j\|\sum -\tilde{a}_j$	多项式时间可解（第7节）
$1\|\sigma_{SYS} = \sigma_T, u_j = b_j - \tilde{a}_jC_j\|\sum -\tilde{a}_j$	多项式时间可解（第7节）

表4：表2中问题7–16的结果概览

本节中，我们考虑给定且固定参数下问题的一般情形，即表2中的前六个问题。具体而言，我们首先提出一种通用的解法，可应用于问题 $1|u_j|u_{\min}$，适用于任意连续效用函数u_j；随后，我们通过引入一种适用于该问题多个版本的（更快的）贪心算法，对其加以改进。最后，我们考察若干变体，包括带释放日期、处理时间相同以及带交期的情形。对于每种变体，我们都给出相应的复杂性结果和求解算法。

4.1 通用解法

给定最优解值 $u_F = u_{\min}$（σ_F）的下界 u 和上界 $\bar{u}$，我们可以采用二分搜索描述一种通用解法（假设效用函数连续）。一般的上界为 $\bar{u} := \max_j\{u_j(0)\}$，而下界可能取决于其他问题数据，例如交期和释放日期。对于线性效用函数，我们可以选择上界 $\bar{u} := \max_j\{b_j\}$。我们可以进行 $\log(\bar{u} - u)$ 次迭代的二分搜索，每次迭代都有一个目标值 u^T。

为了检验 $u_F \ge u^T$，每个作业 j的完工时间 C_j必须满足 $u_j(C_j) \le u^T$，也就是说，回顾公式（1），$C_j \le u_j^{-1}(u^T)$，这意味着一个交期

$$d_j^T = u_j^{-1}(u^T) \tag{5}$$

适用于每个作业。若底层带交期的调度问题可以通过某种已知算法在多项式时间内求解，那么这便可立即转化为我们问题的最优多项式时间算法。事实上，只需证明该带交期问题存在一个没有任何迟交作业的可行解即可。

这类算法的运行时间是多项式级的，但并非强多项式级，因为其中包含 $\bar{u}$ 的对数因子，而该因子取决于编码后输入的长度。因此，对于若干问题变体，我们将介绍新的、更简单的算法，其运行时间为强多项式级，即仅取决于输入值的数量。

4.2 贪心型算法

通过第 4.1 节的二分搜索方法，只要公式（5）中的交期能够在多项式时间内计算出来，同样可以在多项式时间内找到最优解。实际上，可以按交期的非递减顺序对作业进行排序（EDD规则），从而计算最大延迟，并高效检查 $u_F \geq u^T$ 是否成立。不过，我们可以通过以下强多项式贪心型算法避免二分搜索，我们将其称为 MaxMinGreedy。

对于所有未调度作业 T 的完工时间，初始设为 $\sum_j p_j$，我们选取在时间 T 完工时效用最高的作业k 。该过程逐步迭代，每一步都考虑时间 $T := T - p_k$。它适用于任意非递增效用函数 u_j。图1展示了MaxMinGreedy 应用于一个包含三个作业且具有线性效用的示例。请注意，这种方法与*Lawler*算法颇为相似。Lawler算法以在优先约束下最小化正则函数的最大值而闻名（Lawler，1973年）。还要注意，与Lawler算法一样，我们的方法也可以轻松扩展到作业之间存在优先关系的情形。

算法 1 Max-Min Greedy

1：输入：代理作业集合 $J = \{J_1, J_2, \ldots, J_n\}$**2：输出：排列（调度）** $\pi = \pi_1$**，...，** π_n **（**π_i**是序列中的第** i**个作业 3：** T**：**$= \sum_j p_j$**4：对于** $i = n$ **从大到小遍历至 1，执行 5：选取剩余作业中在时间** T **效用最高的作业** $k \in J$**，并将其安排在位置** i**（即** π_i**：**$= k =$**arg max** $\{u_j(T)$**：** $j \in J\}$**）6：** T**：**$= T - p_k$**7：** J**：**$= J \setminus \{k\}$**8：结束循环 9：返回：** π

总之，我们得到以下结果，其简洁的证明见附录。

定理2 如果对于所有 $j \in J$， u_j 都是作业 j的完工时间 C_j 的任意非递增函数，那么 MaxMinGreedy 可在 $O(n^2)$ 时间内计算出 $1|u_j|u_{\min}$ 的最优解。

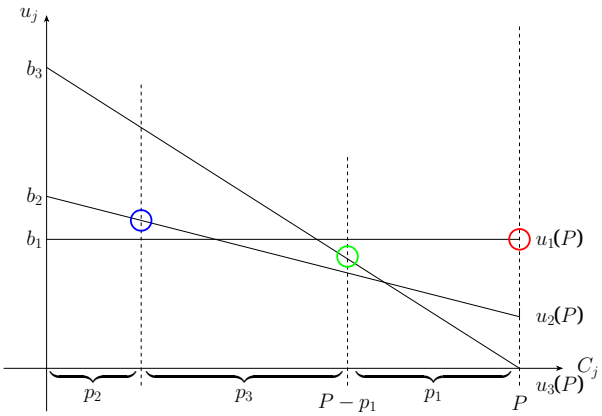

图1：算法 MaxMinGreedy应用于一个包含三个作业和线性效用函数的 $1|u_j|u_{\min}$ 实例。

4.3 释放日期

本节讨论存在释放日期时的问题，即 $1|r_j,\ u_j|f$。我们将探讨每个作业 都有任意给定释放日期的情形，以及除一个作业 外所有作业的释放日期均为零这一特殊情形。

公平解。我们首先关注公平目标，即 $f = u_{\min}$。下文将证明，引入释放日期会使问题变为强 NP 困难，即使所有效用都是线性的也是如此。该结果通过将效用目标值归约为一个等价的交期形式得出。详细证明见附录。

定理 3 问题 $1|r_j, u_j|u_{\min}$ 是强 *NP* 困难，即使效用函数为线性效用函数也是如此。

还可以证明，即使只有一个作业的释放日期严格大于零，该问题仍然是 NP 困难的。在这一受限情形下，该问题实际上是弱 NP 困难的。此结论的证明见附录。

定理 4 问题 $1|r_j, u_j|u_{\min}$ 是弱 *NP* 困难，即使效用函数为线性效用函数且只有一个释放日期非零也是如此。

为说明这一特殊情形确实只是弱 NP 困难，我们简要介绍一种伪多项式动态规划算法。

设作业 $n+1$ 的释放日期为 r，而所有其他作业从开始时起即可处理。设 $P = \sum_{j=1}^{n} p_j$ 为除作业 $n+1$ 外所有作业的总处理时间。我们遍历作业 $n+1$ 的所有可行开始时间 s_{n+1}，即 $s_{n+1} = r,\ r+1,\ \ldots\ldots,\ r+P$。对于每个开始时间 s_{n+1}，我们执行类似于第 4.1 节所述的二分搜索。对于每个候选值 u^T，根据公式（1），我们为每个作业 j 确定交期 $d_j^T = u_j^{-1}(u^T)$，并检查是否存在一种可行调度，使所有作业都不晚于这些交期完成，即最大延迟为 ≤ 0。为此，我们必须决定每个作业是安排在作业 $n+1$ 之前、并在 s_{n+1} 之前完成，还是安排在作业 $n+1$ 之后、且不早于$s_{n+1}+p_{n+1}$开始。在由此得到的两个子集中，作业必须按 EDD（最早

算法2 特殊情形下的伪多项式动态规划算法

$1|r_j,\ \ u_j|u_{\min}$ 仅有一个非零释放日期

输入：作业**1**，.....，$n+1$，处理时间为 p_j，释放日期为 r，对应作业 n+1

输出：最大最小效用 $u^*_{\min}$

令 $P:=\sum_{j=1}^{n} p_j$, $u_F:=0$

对于 $s_{n+1}=r$ **到** $r+P$，**执行**

  对 u^T 执行二分搜索

  当 存在候选值 u^T **时**

    $\mathcal{S}\leftarrow\emptyset$

    计算所有作业 j 的交期 $d_j^T=u_j^{-1}(u^T)$

    识别满足 $d_j^T\le s_{n+1}+p_{n+1}+p_j-1$ 的作业集合 A，集合 $P_A:=\sum_{j\in A}p_j$

    如果 $P_A\le s_{n+1}$，**则**

      $\mathcal{S}\leftarrow\{(P_A,\ 0)\}$

      按 d_j^T 的非递减顺序对剩余作业排序

      对于所有 作业 j 按排序顺序 **执行**

        $\mathcal{S}'\leftarrow\emptyset$

        对于所有状态（$P_1,\ P_2$）$\in\mathcal{S}$ **执行**

          如果 $P_1+p_j\le s_{n+1}$，**则**

            将（$P_1+p_j,\ P_2$）添加到 $\mathcal{S}'$

          结束条件判断

          如果 $d_j^T\ge s_{n+1}+p_{n+1}+P_2+p_j$，**则**

            将（$P_1,\ P_2+p_j$）添加到 $\mathcal{S}'$

          结束如果

        结束循环

        $\mathcal{S}\leftarrow\mathcal{S}'$

      结束循环

    结束如果

    如果 $\mathcal{S}=\emptyset$，**则**

      选择较小的候选项 u^T

    else

      选择较大的候选项 u^T

    如果结束

  当循环结束

  u_F:=最大值$\{u_F,u^T\}$

结束循环

返回 u_F

按交期顺序排列，以最小化最大延迟。所有满足 $d_j^T \le s_{n+1}+p_{n+1}+p_j-1$ 的作业都必须安排在 $n+1$ 之前，因此记录其总加工时间 P_A 后，便可不再考虑它们。对于所有剩余作业，我们采用动态规划，其中每个状态（P_1，P_2）由作业组成：总加工时间分别为 P_1和 P_2的作业，分别安排在作业 $n+1$之前和之后。初始化单个状态（P_A，0）后，我们按 EDD顺序依次处理作业。将每个作业 j 加入每个现有状态（P_1，P_2）时，可以将其加入第一个或第二个子集。因此，状态（P_1，P_2）会被两个新状态（P_1+p_j，P_2）和（P_1，P_2+p_j）取代，前者在 $P_1+p_j \le s_{n+1}$ 时生成，后者在 $d_j^T \ge s_{n+1}+p_{n+1}+P_2+p_j$时生成。显然，状态总数不超过 P^2。如果遇到某个作业无法加入任何现有状态，就可以确定：不存在能使作业 $n+1$的开始时间为 s_{n+1}、且达到所需效用目标的解，因此必须降低 u^T。否则，通过加入最后一个作业生成的每个状态都是可行解，其最小效用至少为 u^T，于是我们可以尝试提高 u^T。上述过程的具体说明见算法2中的伪代码。

总体伪多项式运行时间为 $O(P \cdot nP^2 \cdot \log(UB-LB))$，其中 UB和LB 分别是目标值的上界和下界。

系统最优解。下面我们考虑具有释放日期和全局目标函数的问题 $1|r_j,\ u_j|\sum_j u_j$，即考虑总效用最大化。事实证明，该问题与最大公平性目标的问题具有相同的复杂性。由于完工时间可以映射为效用，因此，从强 NP 困难问题 $1|r_j|\sum Cj$ 进行简单归约即可证明以下结论。

定理5 问题 $1|r_j, u_j|\sum_j u_j$ 是强 *NP* 困难的，即使效用函数是线性的也是如此。

定理6 问题 $1|r_j, u_j|\sum_j u_j$ 是弱 *NP* 困难的，即使效用函数是线性的且只有一个释放日期非零也是如此。

证明。我们采用从 PARTITION 出发的归约。考虑一个实例 I_P，其中包含整数$w_1,\ldots,w_n$和 $B=\frac{1}{2}\sum_{j=1}^n w_j$。相应地，我们构造一个实例 I，属于 $1|r_j,u_j=b_j-a_jC_j|\sum_j u_j$，其中有 $n+1$ 个作业。对于每个 $j=1,..,n$，令 $p_j=a_j=w_j$和 $r_j=0$，并令作业 $n+1$ 的处理时间为 $p_{n+1}=\bar{p}$, $a_{n+1}=M$（其中 M 足够大），释放日期为$r_{n+1}=B$。由于 $\sum_{j=1}^n p_j=2B$，任何可行调度的最大完工时间至少为 $2B+\bar{p}$。对于满足 $n+1$ 在区间 $[B,B+\bar{p}]$ 内处理且最大完工时间等于 $2B+\bar{p}$ 的 I 的可行调度，我们称之为平衡 调度（见图2中的调度 $\tilde{\sigma}$）。我们证明，I_P是一个是实例，当且仅当 I 的某个最优调度是平衡的。

此后，对于任意 $S\subseteq\{1,\ldots,n\}$，我们使用 $p(S)=\sum_{j\in S}p_j=\sum_{j\in S}w_j$。

假设 I 存在一个平衡的最优调度 σ。根据平衡调度的定义，存在一个作业集合 $S\subseteq\{1,\ldots,n\}$，它们在 $[0,B]$中执行，并且 $p(S)=B$。因此，I_P是一个是实例。

反过来，假设 I_P 是实例。于是存在一个子集 $S\subseteq\{1,\ldots,n\}$，满足 $p(S)=B$，且存在一个可行调度 σ，其中 S 中的作业在区间 $[0,B]$内处理，$\bar{}$个作业 $n+1$ 位于 $[B,B+\bar{p}]$中，而剩余作业的集合 S 位于 $[B+\bar{p},2B+\bar{p}]$中。因此，这样的调度是平衡的。注意，由于 $a_j=p_j$，在 S 内或在

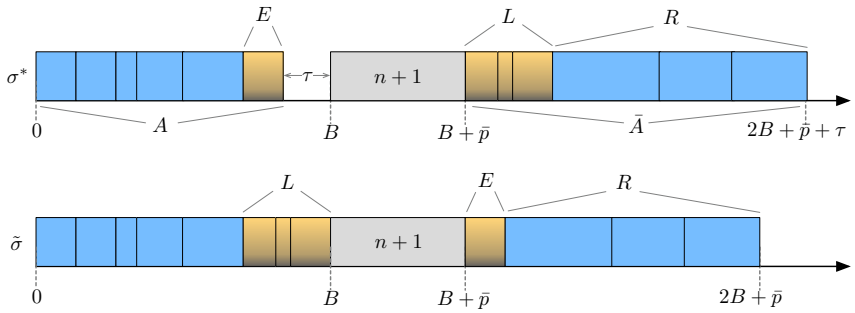

图2：定理6的证明：非平衡调度 σ^* 和通过交换作业块 E 与 L得到的平衡调度 $\tilde{\sigma}$ 。

S 内对作业进行任何内部排列都不会改变总效用。此外，利用标准的交换论证，很容易证明所有平衡调度的总效用都相同。

现在证明，如果存在平衡调度，那么它就是 I的最优调度。同样，假设 I_P 是一个是实例，因此存在一个适用于 I的平衡调度 σ；考虑一个适用于 I 的非平衡最优调度 σ^* 。设 A 和 A 分别为在 σ^*中作业 $n+1$ 之前和之后处理的作业集合。由于 $a_{n+1} = M$ 非常大， σ^* 会尽可能早地安排 $n+1$开始，即在其释放时间 $r_{n+1} = B$开始，并且 $[0, B]$中存在长度为$\tau \geq 1$ 的空闲时间。

由于存在平衡解，因此必定存在两个作业子集 $E \subset A$ （可能为空）和 $\emptyset \neq L \subset A$，使得 $p(L) = p(E) + \tau$。不失一般性（必要时可在 A 和 A 中重新排列作业，且不损失效用），我们可以假设 E 和 L 分别安排为 A中最后一组连续作业，以及 A中最先的一组连续作业。令 R 为在 L 之后处理的剩余作业集合（即 $R = A \setminus L$）。现在，考虑通过交换作业块 E 和 L ，并将 R 中的所有作业提前 τ 而得到的调度方案 $\tilde{\sigma}$ 。那么 $\tilde{\sigma}$ 的最大完工时间为 $2B + \bar{p}$ ，因此它是平衡的。图2展示了这两个调度方案σ^*和 $\tilde{\sigma}$。

将 σ^* 转换为 $\tilde{\sigma}$ 时总效用的变化为

$$\Delta u = \underbrace{-p(E)\big(\bar{p} + \tau + p(E)\big)}_{\text{loss for postponing } E} + \underbrace{p(L)\big(\bar{p} + p(L)\big)}_{\text{gain for advancing } L} + \underbrace{p(R)\tau.}_{\text{gain for advancing } R}$$

由于 $p(L) = p(E) + \tau$，通过简单的代数运算可得

$$\Delta u = \big(p(E) + \bar{p} + \tau + p(R)\big)\tau > 0.$$

因此 $u(\tilde{\sigma}) > u(\sigma^*)$，这与 σ^*的最优性相矛盾。总之，如果 I_P 是是实例，则 I 的最优调度是平衡的，证明至此完成。□

与上一节讨论的公平目标情形类似，当我们聚焦于只有一个作业具有非零释放日期这一特殊情形时，问题 $1 \mid r_j,\ u_j \mid \sum_j u_j$ 是弱 NP 困难的（与之相反，一般情形是强 NP 困难的；参见定理6）。这是因为可以设计一种伪多项式动态规划算法

与上文为 $1 \mid r_j,\ u_j \mid u_{\min}$概述的算法类似。

该算法遍历作业 $n+1$ 的所有可行开始时间 S。它是唯一具有正释放日期 r 的作业。对于每个候选开始时间 s_{n+1}，我们必须决定每个作业是在 s_{n+1} 之前还是之后进行调度。在由此得到的两个子集中，作业都必须按照 $\frac{a_j}{p_j}$ 的非递增顺序进行调度，这与表达式（4）一致。动态规划算法按照此顺序考虑作业，其处理方式与情形 $1 \mid r_j,\overline{\ }u_j \mid u_{\min}$ 类似。算法从单个初始状态 (0，0) 开始，逐个处理作业。对于每个作业 j，任何已有状态 $(P_1,\ P_2)$ 都会被两个后继状态 $(P_1+p_j,\ P_2)$ 和／或 $(P_1,\ P_2+p_j)$ 取代。处理完所有作业后，保留总效用最大的状态，作为与候选值s_{n+1}相对应的最佳解。最优性由作业的规定排序保证。

单位或相同处理时间。调度文献中一个经过深入研究的特殊情形（Baptiste, 1999, 2000; Chen 等人, 2026; Lawler, 1978; Simons, 1978）考虑所有处理时间都相同的实例，即对于所有作业 j，$p_j = p$；或者考虑限制更严格的情形，$p_j = 1$。需要注意的是，这两个假设并不等价。下文将证明，这两种特殊情形 $1|p_j = 1,\ r_j,\ u_j|u_{\min}$ 和 $1|p_j = p,\ r_j,\ u_j|u_{\min}$ 都存在多项式时间算法，尽管其对应最优解的结构性质有所不同。

当所有处理时间均为单位时间时，$1|p_j = 1,\ r_j,\ u_j|u_{\min}$ 的任意解都可以仅通过指派每个作业到一个单位时间槽来表示，该时间槽对应作业的完成时间/开始时间。显然，该时间槽必须可行，即每个作业 j 只能在其（整数）释放日期r_j 与对应调度最大完工时间的时间槽之间进行调度。

在寻找 $1|p_j = 1,\ r_j,\ u_j|u_{\min}$ 的最优解时，显然我们也可以将注意力限制在最大完工时间最短且避免不必要空闲时间的半主动调度方案上。请注意，最小最大完工时间值 T 可以轻松计算出来——如下所示（参见算法 3），任意半主动调度方案中作业可用的时间槽也可以轻松计算。

这一性质使得我们能够采用一种直接的算法方法：调度问题的解对应于作业集合与最小最大完工时间调度所确定的时间槽之间的二分图匹配，其中每个作业只能指派到不早于其释放日期的时间槽。将作业 i 指派到时间槽 C_j 的收益为$u_i(C_j)$。因此，求解 $1|p_j = 1,\ r_j,\ u_j|u_{\min}$ 可归结为线性瓶颈指派问题，该问题可在 $O(n^{2.5}/\sqrt{\log\ n})$ 时间内求解（参见 Burkard 等人, 2012年，第 6.2 节）。

———

我们可以基于第 4.2 节中介绍的算法MaxMinGreedy所采用的相同原理，设计一种效率更高的过程。该过程详见算法 3。首先，按照释放日期的非递减顺序对作业进行排序，然后构造一个最大完工时间为 T 的半主动调度：每个作业一旦到达其释放日期，且排序中所有前序作业均已完成，就立即开始执行（参见算法 3 的第 5–7 行）。

然后，从最后一个可用时间槽开始并迭代执行（第 10–18 行），算法会在能够移至序列末尾且不会增加最大完工时间的作业中，找出在该时间槽效用最高的作业 j'

算法 3 $1|p_j = 1$的精确算法，$r_j,\ u_j|u_{\min}$

1: **输入：** 作业集合 $J = J_1,\ J_2,\ \ldots, J_n$，释放日期为 $r_1,\ r_2,\ \ldots, r_n$
2: 按释放日期的非递减顺序对作业进行排序并重新命名，即 $r_j \leq r_{j+1}$对于 $j = 1, \ldots, n-1$
{对于所有连续子集，此排序仍然有效}
3: $pred(j+1) := j$ {定义当前作业序列}
4: $C := 0$
5: **对** $j = 1$ 至 n **执行**
6: $C := C_j :=$ 最大值$\{r_j + 1, C + 1\}$
7: **结束循环**
8: $T := C_n$
9: $J' := J$ {未调度作业集合}
10: 令 $last$表示 J'中最后一个作业的索引
11: **重复**
12: $j'' := j' := last$ {j' 是最后一个进行调度的候选作业}
13: **当**$C_{j''} - 1 > r_{j''}$ **时执行**
14: $j'' := pred(j'')$
15: **如果** $u_{j''}(T) > u_{j'}(T)$ **则**
16: $j' := j''$
17: **结束条件判断**
18: **结束循环**
19: 使用 $C_{j'} := T$ 将 j' 调度在位置 $|J'|$
20: $J' := J' \setminus \{j'\}$
21: 将所有作业 $\ell \in J', \ell > j'$, 提前 1，并更新 C_ℓ, $pred$, 和 $last$
22: $T := C_{last}$ {如适用，标记新作业块的终点}
23: **直到** $|J'| = 1$
24: 将剩余作业安排在位置 1 处 J'

。只有在这种移动可行时才会执行，也就是说，只有当前序列中所有排在 j' 之后的作业都能提前时，才会执行。只有当这些作业的开始时间均不恰好等于其释放日期时，才能进行这种调整。while 循环会终止，因为第一个作业总是在其释放日期开始。找到这样的作业 j'后，将其移至序列末尾（第 19 行），并相应更新剩余作业 J'及其相关参数（第 20–22 行）。如果原始序列中存在空闲时间，则会隐式处理由此产生的每个作业块；请注意，这类作业块总是以一个恰好在其释放日期开始的作业开头。显然，算法 3 的运行时间为 $O(n^2)$。因此，我们得到以下结果。

定理 7 具有释放日期和单位处理时间的问题 $1|p_j = 1, r_j, u_j|u_{\min}$ 可在 $O(n^2)$ 时间内求解。

与前一种情况一样，很容易证明，在单位处理时间和一般效用函数下确定系统最优解，可以直接重新表述为线性和指派问题。因此，以下结论成立：

观察 8 具有释放日期和单位处理时间的问题 $1|p_j = 1, r_j, u_j|\sum u_j$可在多项式时间内求解。

通过缩放，同样的推理也适用于 $p_j = p > 1$ *if* p 是所有释放日期的公因子 这一情形。然而，对于处理时间 $p_j = p > 1$ 固定但取值任意的情形，这一结论并不成立，下文将对此进行讨论。

$1|p_j = 1, r_j, u_j|u_{\min}$ 其中 $u_j(C_j) = b_j - C_j$ 对于给定常数 $b_j \in \mathbb{Z}+, j = 1, \ldots, n$ 的特殊情形，等价于经典调度问题 $1|p_j = 1, r_j|L_{\max}$。对于整数释放日期，使用所谓的可抢占EDD规则，可以在 $O(n \log n)$ 时间内高效求解该问题（见 Lenstra 和 Shmoys，2020年，第3.2节）。在同一篇论文中，作者进一步指出，所有处理时间 p_j 均等于某个任意正整数 p 的情形，与所有作业均满足 $p_j = 1$ 的情形有本质区别。尽管可以重新缩放时间使得 $p = 1$，但这一变换通常会使释放日期变为有理数。这违反了声称可抢占EDD规则能够求解问题 $1|p_j = 1, r_j|L_{\max}$ 时所依据的前提。

对于我们的问题，也存在类似的情况。事实上，如果所有作业的处理时间都相等，但不等于单位时间，那么推迟已释放的作业，从而产生“非强制性”空闲时间，可能是有意义的。因此，最优调度可能不是半主动调度。图3通过一个包含两个作业、数据如下的示例说明了这一点：$r_1 = 0, r_2 = p/2, u_1 = 3p - C_1, u_2 = 2p - C_2$。最小最大完工时间为 $C_{\max} = 2p$，其实现方式是先安排作业1，再安排作业2，后者于时间 p开始。对于这一调度，有 $u_1 = 2p$ 和 $u_2 = 0$。若先在时间 $r_2 = p/2$ 安排作业2，之后再安排作业1，即于时间 $3p/2$开始，则机器从时间0到 $p/2$ 处于空闲状态，尽管作业1已经释放。这样得到 $u_1 = u_2 = p/2$，其目标函数值更优。

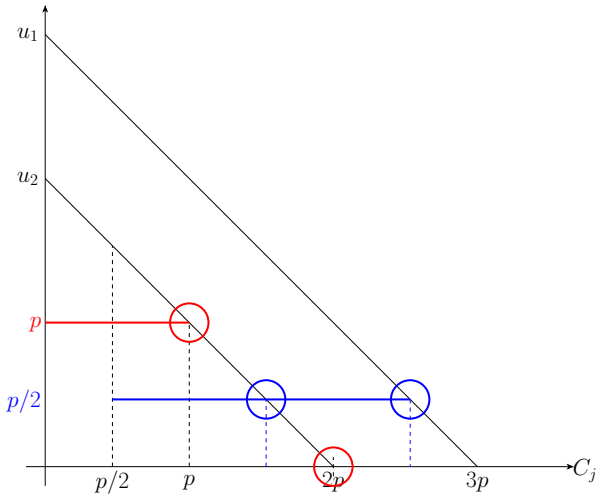

图3：用于说明空闲时间可能有用的示例。红色表示不含空闲时间的调度 $\langle 1,\ 2\rangle$，蓝色表示含空闲时间的最优调度 $\langle 2,\ 1\rangle$。

因此，上述迭代贪心方法不再适用，因为它依赖于已知所有作业的总完工时间。不过，通过对目标函数值进行二分搜索，并利用文献中的一项结果来求解由此产生的子问题，仍可获得多项式时间解。具体而言，Carlier 针对具有释放日期和相等处理时间的最小化延迟作业总数问题设计了一种算法 $1|p_j = p,\ r_j|\sum U_j$，其运行时间为 $O(n^3 \log n)$（Baptiste，1999；Carlier，1981）。因此，对于可行目标值范围内给定的目标值 u^T，我们可以反复应用 Carlier 的算法，并按照表达式 (5) 定义作业的交期，然后

将其与第 4.1 节中介绍的通用二分搜索框架相结合，以解决我们的问题变体。综上，可得以下结果。

定理9 问题 $1|p_j = p, r_j, u_j|u_{\min}$ 可在多项式时间内求解 $O(n^3 \log n \log(UB - LB))$ 其中，UB 和 LB 分别是目标值的上界和下界。

请注意，所讨论的问题可以重新表述为一个标准调度问题：在存在释放日期且处理时间相同的情况下，最小化正则目标函数的最大值。具体而言，上述结果推广了 Lenstra 和 Shmoys（2020年，第3.3节）的研究；他们探讨了最小化最大延迟这一特例，即 $1|p_j = p, r_j|L_{\max}$。他们提出了一种运行时间为 $O(n^3 \log^2 n)$ 的算法，但该算法仅适用于 $L_{\max}$ 目标。他们的方法依赖于一种更复杂的技术，即反复应用最早交货期（EDD）规则，从而将问题分解，并针对该目标高效求解。

关于相应的系统最优解 σ_{SYS}，Baptiste（2000）提出了一种 $O(n^5)$ 动态规划算法，用于求解具有释放日期且处理时间相等的单机调度问题，以最小化总加权完工时间。由此可得：

观察10 问题 $1|p_j = p, r_j, u_j = b_j - a_j C_j| \sum u_j$ 可在多项式时间内求解。

4.4 交期

现在我们转而讨论每个作业 j 还关联一个交期d_j的问题。下文使用常用记号 U_j 表示作业 j 是否迟交，其中$U_j = 1$ 表示作业 j 迟交（即 $C_j > d_j$），而 $U_j = 0$ 表示未迟交。

我们首先假设存在一种调度，使所有作业都能按时完成，即 EDD规则生成的调度中没有迟交作业。在这种情况下，整个调度没有空闲时间。对于这种情形，可以采用算法 MaxMinGreedy 的一个简单变体。我们将所有交期视为截止时间，并在每次对应于时间步 T 的迭代中，仅考虑截止时间满足 $d_j \geq T$的未调度作业。该方法的正确性可由定理2证明中所用的相同论证得出，只需将其应用于每个时间步的受限可行作业集合 J 即可。

定理 11 问题 $1|\sum U_j \leq k, u_j = b_j - a_j C_j|u_{\min}$ 在所有作业都能按时完成（即 $k = 0$）这一特殊情形下，是多项式时间可解的。此外，可以在 $O(n^2)$ 时间内计算出一个最优调度。

现在考虑每个可行调度都必然包含延迟作业的情形。在这种情况下，我们可以假设所有迟交作业获得相同的效用（严格低于交期前可获得的效用，也可能等于 0）。这种情形可以用一个非递增（不一定连续）的效用函数来表示，因此定理2的结论仍然适用。

接下来，我们考虑更为常见的交期设定：所有迟交作业都会被舍弃，也就是说，它们不会为目标函数贡献任何效用。为避免过度“浪费

作业”，我们对迟交作业的数量设定上界 k （等价地，对必须按时完成的作业数量设定下界 $n-k$ ）。显然，只有在 $k \geq U^*$ 时，这项要求才有意义；其中 U^* 表示使用 Moore 算法（Moore，1968年）可得到的迟交作业最少数量。现在，我们可以采用第 4.1 节所述一般方法的一种变体。给定目标效用值 u^T，对于每个作业 j，我们确定作业 j 在仍能按时完工且效用至少达到 u^T 的情况下，最晚的完工时间。据此，我们定义更新后的交期 $d_j^T := \min\{d_j, u_j^{-1}(u^T)\}$，这意味着，任何在时间 d_j^T 前完工的作业都既能按时完成，又能满足目标效用要求。

然后，我们将 Moore 算法（运行时间为 $O(n\log\ n)$）应用于交期经修改的实例 d_j^T，得到 k^T 个迟交作业。如果 $k^T > k$，我们就降低目标效用 u^T；而如果 $k^T \leq k$，则可以提高目标效用。因此，对于迟交作业数量有上限的问题变体，我们得到以下结果。

定理 12 问题 $1|\sum U_j \leq k, u_j = b_j - a_j C_j|u_{\min}$，即确定在至多 k 个作业可能迟交这一约束下，任一准时作业所能达到的最小效用，可以在 $O(n \log\ n \log(UB - LB)$ 时间内求解，其中 UB 和 LB 分别是目标值的上界和下界。

根据前一个定理，自然会想到一个问题：当目标是最大化总效用时，是否也能得到类似的多项式时间结果？有些出人意料的是，此时的情况与该问题其他版本的情况有根本不同。事实上，虽然至多有 k 个作业迟交时的最小效用问题可以在多项式时间内求解，但寻求系统最优的相应问题却是强 NP 困难。

这个问题等价于在截止时间约束下最小化总加权完工时间。Chen 等人（1998 年）指出，带截止时间的加权完成时间和最小化问题是强 NP 困难的，并将这一结论归因于 Lenstra 等人（1977 年）。然而，Lenstra 等人的论文仅通过从背包问题进行归约，证明了弱 NP 困难性。对于此处考虑的情形，我们给出一个直接证明，表明该问题确实是强 NP 困难的。

定理 13 问题 $1|\sum U_j \leq k, u_j = b_j - a_j C_j|\sum u_j$ 即，在最多允许 k 个作业迟交的约束下，最大化按时完成的作业的总效用，是强 *NP* 困难的。

证明。 为证明该结论，我们将 3-Partition 归约到我们的问题。因此，给定 3-Partition的一个实例 I_{3P}，其中有 $3n$ 个整数 $w_1, \ldots, w_{3n}$，满足$\sum_{j=1}^{3n} w_j = W = nT$且$T/4 < w_j < T/2$，其中 $j = 1, \ldots, 3n$，我们按如下方式构造问题 $1|\sum U_j \leq k, u_j = b_j - a_j C_j|\sum u_j$的一个实例 I。

在实例 I 中，共有 $4n-1+k$ 个作业，允许延迟的作业数量上限为 $k \geq 0$。归约的思路是，前 $3n$个作业（“物品作业”）对应于I_{3P}中的整数，接下来的 $n-1$ 个作业在归约中用作调度的“分隔作业”，而最后一个

k “长作业”作业是在任何可行调度中都会迟交的作业。因此，在实例 I 中，我们有

$$p_j = \begin{cases} w_j & \text{for } j = 1, \ldots, 3n \\ M & \text{for } j = 3n+1, \ldots, 4n-1 \\ L & \text{for } j = 4n, \ldots, 4n-1+k \end{cases}$$

其中，$M > 0$ 和 $L \geq nT + (n-1)M =: E$ 均为给定整数。此外，

$$d_j = \begin{cases} E & \text{for } j = 1, \ldots, 3n \\ (j-3n)(T+M) & \text{for } j = 3n+1, \ldots, 4n-1 \\ L & \text{for } j = 4n, \ldots, 4n-1+k. \end{cases}$$

注意，至多只有一个长作业能赶上截止时间；然而，若出现这种情况，它将是调度中唯一准时的作业，因为 $L \geq E$。这会使该解不可行，因为届时将有超过 k 个作业迟交。因此，在任何可行解中，k 个长作业都将迟交，而所有其他 $4n-1$ 个作业（物品作业和分隔作业）都必须准时。

实例 I 中作业的效用函数定义如下：

$$u_j(C_j) = \begin{cases} H - w_j C_j & \text{for } j = 1, 2, \ldots, 3n \\ 0 & \text{for } j = 3n+1, \ldots, 4n-1+k \end{cases}$$

其中，H 足够大，以确保物品作业在整个调度跨度内的效用均非负（例如，$H \geq \max_j\{w_j\} \cdot E$）。

注意，任何针对 I 的可行调度 σ 都可以视为由 n 个（可能为空的）物品作业块组成的序列 $B_i \subseteq \{1, \ldots, 3n\}, i = 1, \ldots, n$，这些作业块由分隔作业 $n-1$ $3n+1, \ldots, 4n-1$ 隔开，后面接着迟交的长作业：$\langle B_1, 3n+1, B_2, 3n+2, \ldots, 4n-1, B_n, 4n, \ldots, 4n-1+k \rangle$。回想一下，迟交作业和分隔作业对总效用的贡献均为零，因此调度 σ 的目标函数值可以表示为 $\sum_{j=1}^{3n} u_j(\sigma) = 3nH - \sum_{j=1}^{3n} w_j C_j$。因此，通过最小化 $\sum_{j=1}^{3n} w_j C_j$，即可得到 I 的最优解。

令 $W_i = \sum_{j \in B_i} w_j$ 表示作业块 i 的总加工时间，令 s_i 表示其开始时间。我们用 $i \prec j$（分别用 $i \preccurlyeq j$）表示作业 i 在调度 σ 中排在作业 j 之前（分别表示排在其之前或与其相同）。那么：

$$\sum_{j=1}^{3n} w_j C_j = \sum_{i=1}^{n} \sum_{j \in B_i} w_j \Big(s_i + \sum_{\substack{\ell \in B_i \\ \ell \preccurlyeq j}} w_\ell\Big) = \sum_{i=1}^{n} \Big\{ \sum_{j \in B_i} w_j \Big((i-1)M + \sum_{\ell \in \cup_{r=1}^{i-1} B_r} w_\ell \Big) + w_j \sum_{\substack{\ell \in B_i \\ \ell \preccurlyeq j}} w_\ell \Big\} = \quad (6)$$

$$= \sum_{i=1}^{n} \Big\{ \Big(M(i-1) \sum_{j \in B_i} w_j \Big) + \sum_{j \in B_i} \Big(w_j \sum_{\substack{\ell \in \{1, \ldots, 3n\} \\ \ell \preccurlyeq j}} w_\ell \Big) \Big\} = \quad (7)$$

$$= M \sum_{i}^{n} (i-1) W_i + \sum_{j=1}^{3n} w_j^2 + \sum_{j=1}^{3n} \sum_{\substack{\ell=1 \\ \ell < j}}^{3n} w_j w_\ell = M \sum_{i=1}^{n} (i-1) W_i + \frac{1}{2} \Big(\sum_{j=1}^{3n} w_j^2 + W^2 \Big). \quad (8)$$

注意，（8）中的双重求和 $\sum_j \sum_\ell w_j w_\ell$ 等于以下乘积之和 $w_i w_j$

其中 i 和 j 是不同的作业。此外，注意到同一等式中的最后一项 $\text{+}_2(\cdots)$ 与作业块的组成无关，只取决于 w_j 的具体取值。因此，对于给定的实例 I，在 w_j 的取值固定时，判定问题简化为确定作业块的组成 B_i，以使项 $\sum_{i=1}^{n}(i-1)W_i$ 最小。

由于求和式 $\sum_{i=1}^{n}(i-1)W_i$ 中的系数递增，最优解会尽可能将处理时间单位安排在较早的块中，同时确保分隔作业不会迟交。

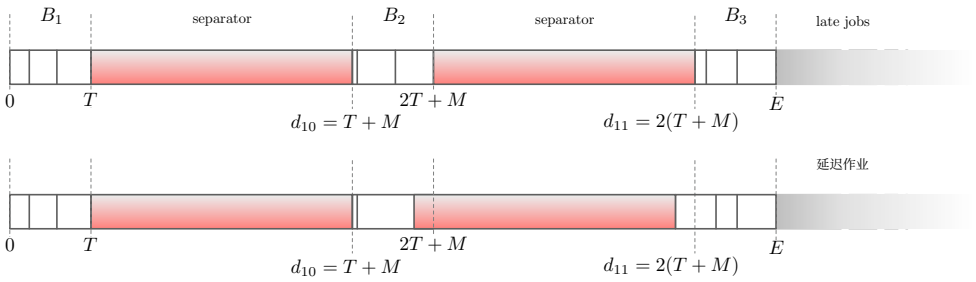

图 4：对应于 a YES-实例（上图）和具有 $n=3$ 的 3-PARTITION NO-实例的最优调度方案。

如果 I_{3P} 是一个 YES-实例，那么我们处于理想配置（如图 4 上方的图表所示），此时可对所有 $i=1,\ldots,n$设置 $W_i=T=\frac{W}{n}$。因此，每个块都会被分配恰好足够的长度，从而使序列中的每个分隔作业都恰好在其指定的截止时间完成。在这种情况下，根据 (8)，在假设 I_{3P} 是一个 YES- 实例的条件下，实例 I 的最优解 σ_{SYS} 的值 u_{YES} 为：

$$\begin{aligned} u_{\text{YES}} &= 3nH-\tfrac{1}{2}\left(\sum_{j=1}^{3n}w_j^2+W^2\right)-M\frac{W}{n}\sum_{i=1}^{n}(i-1) \\ &= 3nH-\tfrac{1}{2}\left(\sum_{j=1}^{3n}w_j^2+W^2\right)-\frac{1}{2}(MW(n-1)). \end{aligned} \tag{9}$$

如果 I_{3P}是一个 NO-实例，那么必定存在一个块 $i\in\{1,\ldots,n\}$，使得块序列由 $W_\ell=T$、对于 $\ell=1,\ldots,i-1$，以及 $W_i<T$组成（参见图 4 下方对应$i=2$的图表）。这意味着必须至少将一个单位的处理时间从块 i 转移到较后的某个块，而该块对应的系数在 $\sum_{j=1}^{3n}w_jC_j$ 中严格更大。由于分配给分隔作业的截止时间，$\sum_{j=1}^{3n}w_jC_j$ 的任何增加都无法在后续块中得到补偿：块超出 T 的量受限于较早的块在 T之前完成所产生的松弛量。因此，与 (9) 中的 u_{YES} 相比，最优效用 u_{SYS} 至少减少 M。

总之，I_{3P} 是一个 YES-实例，当且仅当对应的 $1|\sum U_j\le k|\sum u_j$ 实例 I 的最优效用值 u_{SYS} 不大于 u_{YES}。□

5 可调整的线性效用函数

为了提升系统性能而修改或调整问题参数的想法，已在调度与优化文献中得到广泛研究，其总体框架称为可控加工时间，或更广义地称为可控参数。在这些模型中，可以改变某些作业特征，通常需要付出一定成本或受给定预算的限制，以获得更理想的调度或目标值（Biskup，2008年；Gordon 等，2002年；Huang 和 Wang，2014年；Lu 等，2015年；Gawiejnowicz，2008年；Cheng 等，2006年；Shabtay 和 Steiner，2007年）。

受这一研究方向的启发，我们研究如何通过对各个作业的效用函数进行有控制的调整，改善整体性能指标。在本节中，我们始终只考虑形式为 $u_j = b_j - a_j C_j$ 的线性效用函数，其中每个作业 j 都满足 $a_j \geq 0$，表示随着完工时间增加，估值会降低。我们首先考察对允许修改的总量施加固定全局上界的情形。随后，我们转向一种互补框架：在固定的“面积”约束下调整效用，这反映了必须在各代理之间以平衡的方式分配改进的情形。

在接下来的两节中，我们考虑两种自然的修改方式：提高截距 b_j，以及减小负斜率 a_j。这两种操作都会提高准时作业的效用，我们的目标是提高所有代理所能实现的最大最小效用。由于这类修改不能任意增大，我们假设存在全局预算或容量约束，以限制允许的修改总量。值得注意的是，将公平性替换为效率后，这些问题的自然类比可以相当直接地求解。

5.1 截距的修改 b_j

˜问题 $1|u_j = b_j - a_j C_j|u_{\min}$。首先，我们考虑将每个 b_j 增加一定的 Δb_j，使总增量不超过给定的预算上界 B，即 $\sum_j \Delta b_j \leq B$，其中 $\Delta b_j \geq 0$。

可以使用以下迭代算法进行此类增量调整（技术假设为$a_j > 0$）。从 $\Delta b_j = 0$（对于 $j = 1$、.....、 n）开始，我们将 J定义为当前调度中效用值最低的所有作业的集合。起初，通常有$|J| = 1$，例如 $J = \{j'\}$。（在 $|J| > 1$的情况下，扩展此过程是显而易见的。）我们首先增加 $\Delta b_{j'}$，直到 j' 增加后的效用与某个作业 j''的第二高效用值相等，或者增量达到 $\Delta b_{j'} = B$，此时停止。否则，将 j'' 加入 J。接着，对所有 $\ell \in J$ 的 Δb_ℓ 统一增加，直至增加后的效用达到不属于 J的任一作业的最低效用，此时将该作业加入 J；或者直至 $\sum_{\ell \in J} \Delta b_\ell = B$。后一种情况会结束该过程；若为前一种情况，则继续迭代。

请注意，这一修改过程不会改变 MaxMinGreedy 最初给定的作业序列。考虑任意一个效用有所增加的作业 j' $\Delta b_{j'}$，即可看出这一点。对于任何满足 $C_i < C_{j'}$ 的作业 $i \notin J$，算法都不会在时间 $C_{j'}$ 将 i 纳入调度，因为 $u_i(C_{j'}) \leq u_{j'}(C_{j'})$。显然，增加 $\Delta b_{j'}$ 只会进一步强化这一条件。对于任何满足 $C_k > C_{j'}$ 的作业 $k \notin J$， $\Delta b_{j'}$ 的增加最迟会在达到 $u_{j'}(C_{j'}) = u_k(C_k)$ 时停止。由于效用函数在此后也呈递减趋势，这次增加之后我们仍有 $u_k(C_k) \geq u_{j'}(C_k)$，且决策

MaxMinGreedy 保持不变。[1] 将作业 j' 与 J 中的任意作业进行比较，可以清楚地看出，两个作业的效用都以相同幅度增加，因此对于任何完工时间，它们的效用差异都保持不变。

这种方法不仅适用于线性效用函数，也适用于任意效用函数，此时给定的效用函数 $u_j(C)$ 会加上增量 Δb_j 。相应的过程将在定理 16 的证明中使用。

虽然上述过程可以在 $O(n^2)$ 时间内轻松完成，但我们还可以做得更好。给定一个作业序列，问题实际上是要找出一个休息时段作业 j_b，使得对于所有满足 $u_j \leq u_{j_b}$ 的作业 j ，将截距 b_j 增大，直到它们达到至少为 u_{j_b} 的共同效用值；而所有满足 $u_j > u_{j_b}$ 的作业 j 则保持不变。这个任务可以通过线性时间中位数算法的一种变体在 $O(n)$ 时间内解决（以线性时间求解背包问题中的划分项时采用相同的方法，见 Kellerer 等人，2004，第 3.1 节）

需要注意的是，如果我们也允许 $\Delta b_j < 0$ ，那么问题最终总会得到一个所有效用值都相等的解。这个共同效用值的水平仅取决于 B。假设情况并非如此，我们可以降低任意一个效用高于当前最小效用的作业 j 的值 Δb_j ，并利用预算约束中因此产生的结余，提高所有 $\ell \in J$ 的 Δb_ℓ 值，从而改善最小效用。同样在这种情况下，降低某些 b_j 值不会影响作业序列。对于在作业 j 之后完成的作业，这显而易见；对于在 j 之前完成的作业，也很清楚，因为 j 被选为效用最高的作业。如果降低它的值会使其与较早调度的作业持平，那么由于效用函数非递增，这些作业在当前位置的效用更高，因此算法会停止降低 j ，转而处理这些作业。

5.2 斜率的修改 a_j

问题 $1|u_j = b_j - \tilde{a}_j C_j|u_{\min}$。我们将相同的概念应用于斜率 a_j 的降低：考虑满足 $\sum_j \Delta a_j \leq A$ 的值 $\Delta a_j \geq 0$ ，并由 $u_j = b_j - \tilde{a}_j C_j = b_j - (a_j - \Delta a_j) C_j$ 给出修改后的效用。为保持效用非递增这一性质，我们将施加条件 $\Delta a_j \leq a_j$。

我们可以给出一种与上述情形相当类似的迭代算法，在上述情形中我们选择了 Δb_j 值。和之前一样，我们从调度中效用值最低的作业 j' 开始，该调度由MaxMinGreedy给出，并增加 $\Delta a_{j'}$ ，直到 j' 增加后的效用达到某个 j''。尽管对于 $\ell \in J$ ， Δa_ℓ 值的增幅并不一致，但仍可采用上述相同的步骤。要达到任何新的共同最低效用值 $\bar{u}$，必须为每个 $\ell \in J$ 设置 $\Delta a_\ell := a_\ell - \frac{b_\ell - \bar{u}}{C_\ell}$。请注意，一旦某个作业 $\ell \in J$ 达到 $\Delta a_\ell = a_\ell$，就可以停止该过程，因为此时作业 ℓ 的效用已无法再提高。

上述论证仍然成立，即这些修改不会影响作业序列。对于作业 i ，这显而易见；对于 k ，上述论证适用于 $b_{j'} \geq u_k(C_k)$ ；而对于 $b_{j'} < u_k(C_k)$ ，显然，作业 j' 永远无法达到 k 的效用，因此 k 永远不会被纳入 J。

与截距的情况类似，我们也可以考虑将问题扩展到允许 $\Delta a_j < 0$ 的情况。如上所述，我们预期会得到一个所有效用

[1] 在特殊情形 $a_{j'} = 0$ 下，调整 j' 和 k 的排序会更有利。

值都相等的解。然而，并非所有情况下都能达到这种状态，因为对于每个为 $\Delta a_j = a_j$ 达到的 j， b_j 都是 u_j（C）的上界。其思路是先提高所有 a_j，直到所有效用都等于 u_F，再将斜率增加所累积的量计入预算 B。然后同时降低所有斜率，从而提高所有作业的共同效用值（每个作业消耗不同的 Δa_j），直到预算耗尽或某个作业达到 $\Delta a_j = a_j$。

5.3 具有固定面积的代理效用修改

如果允许截距 b_j 和斜率a_j 都变化，那么对线性效用函数的形状施加额外限制是合理的。我们不仅要求修改后的效用$\tilde{u}_j$ 在所有不超过 P的完工时间内均保持非负，还对代理 j的期望效用设定上界。

˜ ˜如第3.2节所述，变化后的面积 $A_j = P(b_j - \tilde{a}_j(P/2))$（对于线性函数的每种取值，该面积都构成一个梯形）可解释为代理 j的期望效用回报度量，前提是未知变量 C_j 在区间 $[0, P]$上均匀分布。还要注意，斜率 $\tilde{a}_j$可视为代理 j风险态度的度量。 $\tilde{a}_j$ 值较小（分别较大）表示代理更倾向于风险规避（分别更倾向于风险容忍）。

˜在我们的设定中，我们要求 A_j 不得超过原始期望效用回报 A_j。由于每个代理都会充分利用给定的面积 A_j，即 $\tilde{A}_j := A_j$ 我们有 $\tilde{b}_j = \frac{A_j}{P} + \frac{\tilde{a}_j P}{2}$，同时 $\tilde{a}_j \in [0, 2\frac{A_j}{P^2}]$ 以保持 $u_j \geq 0$。很容易看出，效用函数的每条直线都必须经过中点（$P/2$, A_j/P）。逐个考察作业可知，对于作业 j˜，若其完工时间为 $C_j \geq P/2$，最佳选择是设定 $\tilde{a}_j = 0$ 和 $b_j = A_j/P$。另一方面，对于 $C_j \leq P/2$，三角形能够使效用最大化，即尽可能增大 $b_j = 2A_j/P$并令 $\tilde{a}_j = 2A_j/P^2$，使得 $\tilde{u}_j(P) = 0$。综上，调度 σ 中为作业 j 选择的线性效用函数 $\tilde{u}_j(C)$可表示为

$$\tilde{u}_j(C_j(\sigma)) = \begin{cases} \frac{2A_j}{P}(1 - \frac{C_j(\sigma)}{P}) & \text{if } C_j(\sigma) \leq \frac{P}{2}; \\ \frac{A_j}{P} & \text{if } C_j(\sigma) > \frac{P}{2}. \end{cases} \tag{10}$$

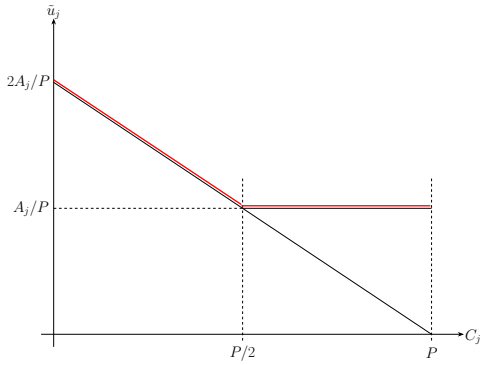

图 5：函数 $\tilde{u}_j(C_j(\sigma))$，如式（10）所定义。

因此，我们可以将每个效用函数表示为两种情形中的较大者：常数~ ~或三角形，如图 5 所示。因此，对于问题 $1|u_j = b_j - \tilde{a}_j C_j,\ A_j = A_j|u_{\min}$ 代理的效用函数随完工时间单调不增，我们可以立即通过应用 MaxMinGreedy得到最优解。事实上，这意味着作业按照面积非递减的顺序进行调度 A_j。

5.4 对单个作业的修改

给定一个 $1|u_j = b_j - a_j C_j|u_{\min}$ 实例及由 MaxMin-Greedy 计算出的一个最优解，我们考虑这样一种情况：单个作业/代理（设为 i）可以修改其效用函数，而其他所有作业保持不变。对于 u_i的每次变动，所得问题的解仍由 MaxMinGreedy计算。

我们首先讨论作业 i 的截距 b_i 可以增加 $\Delta b_i \geq 0$的情况。显然，边际增加 Δb_i 会使 u_i 直接增加相同的量 Δb_i。然而，当 Δb_i 增加到某个程度时，将导致 MaxMinGreedy 改变某个作业 k 的决策；该作业在原调度中的排程晚于 i 。当Δb_i 达到满足以下条件的最小值时，就会发生这种情况：存在一个作业 k，使得 $C_k > C_i$ 且 $b_i + \Delta b_i - a_i C_k = u_k$（$C_k$）。显然， k 可以是任何满足 $C_k > C_i$ 的作业，不一定是 i的后继。

此时，作业 i 占据调度中的位置 k，其完工时间因此变为 C_k，效用也随之骤降 a_i($C_k - C_i$)。此外，先前安排在 i 和 k 之间的作业将由 MaxMinGreedy 重新考虑，并可能采用不同的调度方式，但这不会影响 i。

通过这种方式，我们可以确定修改后的效用函数 $\tilde{u}_i(\Delta b_i)$，它给出作业 i 的效用，截距为 $b_i + \Delta b_i$，这是由 MaxMinGreedy[2]产生的。它是一个斜率为 1 的锯齿函数，每当 i 达到新的完工时间时就会出现不连续点，并在该点取得局部最大值，见图 6。在某个时刻， i 会在调度中被 relegated 到最后位置，且 $C_i = P$，因此 $\tilde{u}_i(\Delta b_i)$会以斜率 1 无限增大。

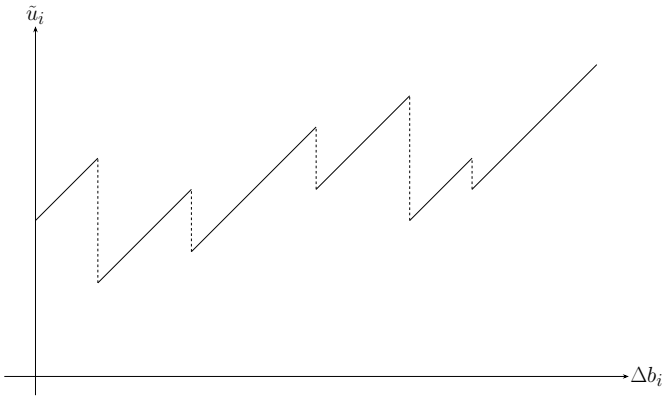

图 6：效用增加量 $\tilde{u}_i$ 随截距增加量 Δb_i变化的关系。

[2]请注意， $\tilde{u}_j(\Delta b_j)$表示效用随截距变化量的变化关系，而 $u_j(C_j)$表示作业的效用取决于其完工时间。

我们关注局部最大值构成的递增序列。显然，只有当提高作业 i 的 Δb_i 能够产生一个高于此前所有已确定局部最大值的新局部最大值时，这种提高才有意义。考虑一个上界 B，并询问 $\Delta b_i \leq B$可达到的最佳效用，也可能是自然的。这要么是该递增序列中在未耗尽 B的情况下得到的最后一个最大值，要么是在 $\tilde{u}_i(B)$处得到。

通过 Δa_i降低负斜率 $-a_i$，也可以采用类似的方法。在这种情况下，每当 i 的完工时间发生变化时，相应的修改后效用函数仍是一个会出现不连续下降的锯齿函数。由于 $\tilde{u}_i(\Delta a_i) = b_i - a_i C_i + \Delta a_i C_i$，该函数的斜率为 C_i，并随着 i 在调度中的每个新位置而增大，如图 7 所示。显然，我们施加上界 $\Delta a_i \leq a_i$，这也会使修改后效用函数达到最大值。

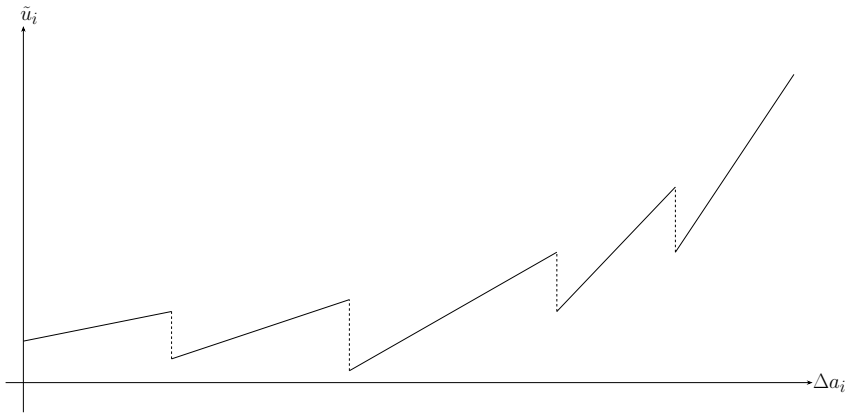

图 7：效用 $\tilde{u}_i$ 随斜率 Δa_i的增大而增加。

修改效用函数的第三种方式，是在假设其他所有代理 j 的面积 A_j保持不变的情况下，将效用函数面积 A_i 增加 ΔA_i。回想一下，MaxMinGreedy 会使作业按 A_j递增的顺序进行简单排序。正如第 5.3 节所讨论的，在 $P/2$处的效用断点意味着，一旦 $C_i \geq P/2$，修改后的效用函数 $\tilde{u}_i(\Delta A_i)$便以斜率 $1/P$线性增长。对于 $C_i < P/2$，我们再次观察到一个锯齿函数，其正斜率为 $\frac{2}{P}(1 - \frac{C_i}{P})$。由于当 $A_i + \Delta A_i$与其他面积之一 A_j相等时，C_i 会增大，因此线性段的斜率会逐渐减小；当 C_i 趋于 $P/2$时，斜率趋近于$1/P$。图 8 展示了这一现象。

6 重新调度以最大化新代理的效用

在本节中，我们考虑一个两阶段优化问题。在第一阶段，针对一组包含 n 个作业的问题，由 MaxMinGreedy 求解含有 n 个代理的标准问题 $1|u_j|u_{\min}$。此步骤得到一个最优调度 σ_F，其为每个作业 j 赋予效用值 $u_j(\sigma_F)$，并得到最大最小效用 $u_F := \min_j\{u_j(\sigma_F)\}$。

在第二阶段，我们考虑一位新代理 $n+1$ 的到来，并要求制定一个新调度 σ，其中包含新代理的作业以及所有原始作业 1、.....、n。新代理具有较高的优先级，应获得最大效用（即应尽可能早地完成），

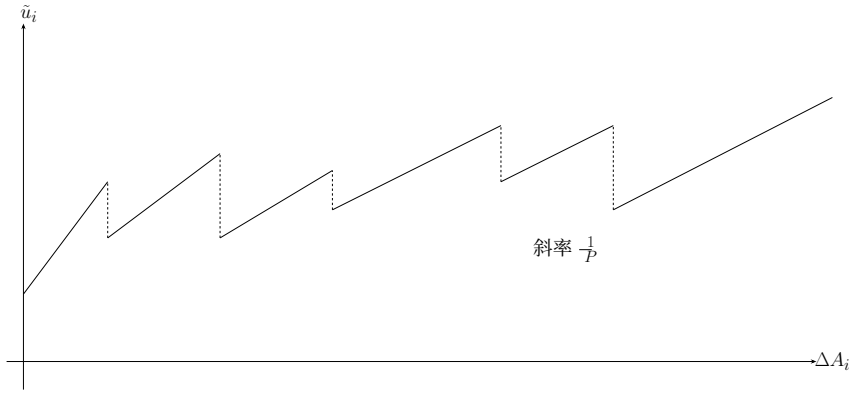

图 8：效用的增加 $\tilde{u}_i$ 与面积的增加 ΔA_i之间的关系。

同时对原始作业的效用施加一定限制。显然，在优先位置安排一项额外作业可能会降低其他作业的效用。因此，我们引入预算R ，可用于补偿代理（作业），方法是提高其效用（或者，更一般地说，维持对其效用的某些限制）。我们将预算支付视为效用增量，并假设只要满足 $\sum_{j=1}^{n} \beta_j \leq R$，就可以向每个代理 j 分配任意非负额度 $\beta_j \geq 0$ ，并将其加到该代理的效用上。

这个问题与此前一些关于重新调度的研究相契合。在这些研究中，必须重新安排某个调度，以便在相对于原调度的变化受到限制的情况下优化给定的目标函数。关于单机重新调度问题的论文，参见例如 Hall 和 Potts（2004）、Nicosia 等人（2021）以及 Rener 等人（2025）。在这些研究中，作者定义了与作业相关的不同扰动 度量，例如原调度与重新排序后的调度之间的位置差异（或完工时间差异）。

由于代理力求最大化自身效用，我们在此将某个代理的扰动 j 度量为新调度 σ （包括可能的补偿）中的效用与原解 σ_F中的效用之差，即:

$$\delta_j = u_j(\sigma) + \beta_j - u_j(\sigma_F). \tag{11}$$

我们考虑重新调度问题的三个合理版本，它们对代理人的扰动设有不同限制:

问题 $1|$重新安排 $\delta_j \geq 0|u_{n+1}$。在这个问题中，我们施加（在某种意义上）最严格的要求：加入新作业后， σ 中的每个单个代理的效用（包括补偿）都应至少与其在 σ_F中的效用一样大。回顾公式（11），我们可以轻松地将约束改写为代理人的扰动，具体如下：$u_j(\sigma) + \beta_j \geq u_j(\sigma_F)$ $\Leftrightarrow \delta_j \geq 0$ 并将问题写为

$$\max_{\sigma} \{u_{n+1}(\sigma) \ : \ \delta_j \geq 0, \ j = 1, \ldots, n\}. \tag{12}$$

问题 $1|$重调度 $\sum \delta_j \geq 0|u_{n+1}$。在这里，我们放宽了前一个问题 (12) 的要求，将效用总和设为上限，而不是分别为每项效用设定上限。同样，我们可以

轻松地将约束改写为代理人的扰动形式，如下所示： $\sum_{j=1}^{n} u_j(\sigma) + R \geq \sum_{j=1}^{n} u_j(\sigma_F)$ $\Leftrightarrow \sum_{j=1}^{n} \delta_j \geq 0$ 由此得到问题的以下形式：

$$\max_{\sigma} \left\{ u_{n+1}(\sigma) \ : \ \sum_{j=1}^{n} \delta_j \geq 0 \right\}. \tag{13}$$

在这种情况下，预算 R 无需在代理之间分配，而是可以直接加到总效用中。

问题 $1|$重新调度 $u_j \geq u_F|u_{n+1}$。在这个问题中，我们要求重新调度的方案 σ 中的公平性值相对于 σ_F没有变差，即所有代理获得的效用都不低于 $u_F = \min\{u_j(\sigma_F)\}$。回顾公式 (11)，我们注意到，在这种情况下，我们是在限制代理人的扰动，如等价关系 $u_j(\sigma) + \beta_j \geq u_F \Leftrightarrow$ $\delta_j \geq u_F - u_j(\sigma_F)$所示。（注意，对于所有 j，$u_F - u_j(\sigma_F) \leq 0$。）最终，我们可以将该问题写为：

$$\max_{\sigma} \left\{ u_{n+1}(\sigma) \ : \ \delta_j \geq u_F - u_j(\sigma_F), \ j = 1, \ldots, n \right\}. \tag{14}$$

我们首先证明，我们的前两个问题版本 $1|$重新调度 $\delta_j \geq 0|u_{n+1}$ 和$1|$重新调度 $\sum \delta_j \geq 0|u_{n+1}$ 都是困难的。

定理 14 问题 $1|resch\ \delta_j \geq 0|u_{n+1}$ 是弱 *NP* 困难，即使在线性效用函数的情况下也是如此。

证明。我们考虑一个由 n 个整数 $w_1, \ldots, w_n$ 定义的实例 $I_P$of Partition，并令 $B = \frac{1}{2}\sum_{i=1}^{n} w_i$。该问题要求找出一个数字子集 S，使其总和为 B。

现在，我们构造问题 (12) 的判定版本的一个实例，其中有 $n+2$ 个具有线性效用函数的原始作业、一个新作业 $n+3$ 和一个预算 R。较大的常数 M 将在稍后设定。通过 MaxMinGreedy 安排这些 $n+2$ 个作业，得到一个调度 σ_F，其效用为 $u_j(\sigma_F)$。然后我们询问，是否存在一个新调度 σ，使作业 $n+3$ 最迟在时间 $2B$ 完成，同时所有其他作业获得效用和支付 β_j，并满足$u_j(\sigma) + \beta_j \geq u_j(\sigma_F)$。

作业的数据列于下表，我们设定 $R = MB + 3B^2$。

j	p_j	a_j	b_j
$1, \ldots, n$	w_j	w_j	$4B(M+2B)$
$n+1$	M	M^2	$\frac{5}{2}M^3$
$n+2$	B	ϵ	$2M^3 - 3M^2B$
$n+3$	B	-	-

我们选择足够大的 M，使得由 MaxMinGreedy 生成的调度由作业 1 到 n按某种未指定的顺序排列，后接作业 $n+1$和 $n+2$，如图9所示的上方调度。为此，我们在时间 $P = 2B + M + B$ 处计算效用，该时间是整体

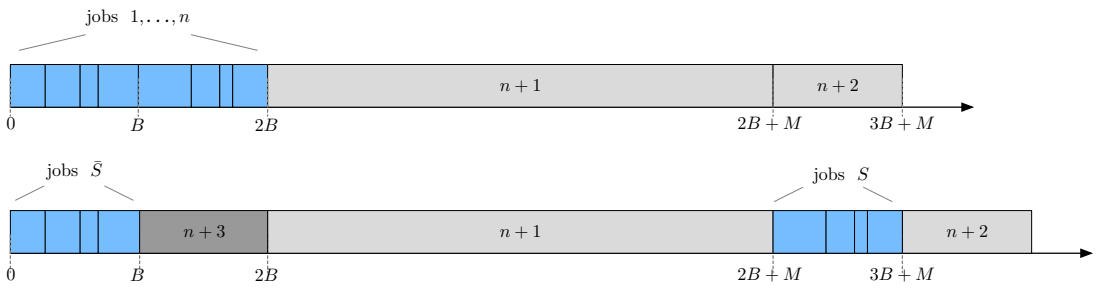

图9：由 MaxMinGreedy 输出的原调度 σ_F （上）以及插入作业 $n+3$ 后的调度（下）。

最大完工时间。我们设定 $M := 5B^2$ ，得到:

$$
\begin{aligned}
u_j(P) &= 4B(M+2B) - a_j P \le 20B^3 + 8B^2 \quad j = 1,\ldots,n,\\
u_{n+1}(P) &= \frac{5}{2}M^3 - M^2(3B+M) = \frac{3}{2}(5^3B^6) - 75B^5,\\
u_{n+2}(P) &= 2M^3 - 3M^2B - \epsilon(3B+M) = 2(5^3B^6) - 75B^5 - \epsilon(5B^2+3B).
\end{aligned}
$$

因此， $n+2$ 在 P 时的效用最大，故安排在最后一个。

在 $n+2$ 之前，即时间 $2B+M$，我们有 $u_{n+1}(2B+M) = \frac{5}{2}M^3 - M^2(2B+M) = \frac{3}{2}(5^3B^6) - 50B^5$，这显然大于 $u_j(2B+M) = 4B(M+2B) - a_j(2b+M) \le 20B^3 + 8B^2 \text{for} j = 1,\ldots,n$。因此， $n+1$ 被安排在 $n+2$ 的正前方。

安排新作业 $n+3$ 的论证如下。插入作业 $n+3$ ，使其开始时间不晚于 B ，需要推迟作业 1 到 n 中的一个子集 S ，且这些作业的总处理时间至少为 B。我们将看到，推迟作业 $n+1$ 会导致支付过高，而作业 $n+2$ 可以推迟到调度末尾，且效用损失最小。因此，我们希望找到一个总长度恰好为 B 的集合 S ，并将这些作业移到作业 $n+2$ 留下的空档中。

显然，完工时间不晚于原先的作业不会降低其效用。然而，每个完工时间更晚的作业，每比原调度晚完工一个时间单位，效用就会损失a_j 个单位。

仅将作业 $n+1$ 的完工时间延长一个时间单位，就需要支付 $M^2 = 25B^4$作为补偿，这超出了预算 $R = 5B^3 + 3B^2$。因此，在 σ中，该作业不能比原先更晚完工。另一方面，可以将作业 $n+2$ 移至新调度的末尾，从而使效用损失最小。

要将作业 $n+3$ 安排在区间 $[0, 2B]$ 中的某处，将作业1、.....、 n 排在作业 $n+3$ 和 $n+1$之间并无意义。由于在调度σ中， $n+1$ 的开始时间为 $2B$ （或更早），我们总可以将 $n+3$ 移至 $n+1$的开始处完工。因此，作业1、.....、 n中的某些作业（记为子集 S）必须移至作业 $n+3$ 和 $n+1$ 之后开始，且其最早开始时间为 $M+B$。集合 $S = \{1,\ldots,n\} \setminus S$中的其他作业则会按原来的序列安排在 σ 的开头，且效用不会降低，见图9中的下方调度。

移动某项作业 $j \in S$ 所需的支付至少应等于其效用最小损失，即$a_j(M + B - (2B - a_j)) = a_j(M - B + a_j)$（最早的新开始时间减去最晚的旧开始时间

时间）。

另一方面，将第一个作业移至最后位置，并使其在时间 $M+3B$、即作业 $n+2$ 开始前刚好完成时，$j \in S$ 的效用损失最大，为 a_j（$M+3B-a_j$）。

显然， $\sum_{j\in S} p_j$ 必须至少为 B，才能容纳作业 $n+3$。对于每个满足$\sum_{j\in S} p_j = B$的子集 S，所需支付总额至多为 $\sum_{j\in S} a_j$（$M+3B-a_j$） $\leq \sum_{j\in S} a_j$（$M+3B$） $=MB+3B^2 = R$, 因此这是一个可行解。然而，对于每个满足 $\sum_{j\in S} p_j \geq B+1$ 的子集 S，所需支付至少为 $\sum_{j\in S} a_j$（$M-B+a_j$） $\geq \sum_{j\in S} a_j$（$M-B$） $\geq$（$B+1$）（$M-B$） $=5B^3+4B^2-B$，这大于 R。因此，当且仅当存在满足 $\sum_{j\in S} w_j = B$ 的子集 S 时，新作业 $n+3$ 才能在完工时间 $2B$ 完成。□

定理 15 问题 $1|resch\ \sum \delta_j \geq 0|u_{n+1}$ 即使在线性效用函数下也是弱 *NP* 困难。

证明。 为证明问题 $1|resch \sum \delta_j \geq 0|u_{n+1}$ 的 NP难性，我们采用 Strusevich 和 Rustogi（2016年）讨论的一种调度问题。他们提出了这样一种单机作业调度问题：调度中必须包含长度为 Γ的休息时段，以便进行某项维护操作。只要不可用时段的开始时间 τ 不晚于给定的截止时间 D_{MP}，就可以自由选择该时段的开始时间。对于总加权完工时间这一目标，该问题记为 $1|\tau \leq D_{MP},\ \Gamma|\sum w_j C_j$，其判定版本要求判断是否存在一个满足 $\sum w_j C_j \leq W$ 的¯调度。Strusevich 和 Rustogi（2016年）证明了该问题是弱 NP 困难。

¯给定一个 $1|\tau \leq D_{MP}, \Gamma|\sum w_j C_j \leq W$判定版本的实例 I_{MP}，我们按如下方式构造本问题判定版本的实例 I，其目标值为 C^T（针对新代理$n+1$）。作业 1 至 n 的处理时间与 I_{MP}中的 n 作业相同，而$p_{n+1} = \Gamma$ 和 $C^T = D_{MP} + \Gamma$。线性效用函数设为$a_j = w_j$ 和任意b_j。为方便起见，令 $\bar{f}$:= $u_{SYS}(\sigma_F)$ $= \sum_{j=1}^{n} u_j(\sigma_F)$。最后，将 I中的预算设为_ _R: $= W+f-\sum_j b_j$。（注意，我们可能有 $R<0$，但这可以通过表示总效用所需的增加量来处理。）

我们将为两个问题考虑作业 1,...,n 的相同调度方案。如果 I是一个 YES实例，其调度为 σ，完工时间为 C_j，那么 $C_{n+1} \leq C^T$，这意味着与 I_{MP}中的维护操作对应的作业 $n+1$最迟在 $C^T - p_{n+1} = D_{MP}$开始，从而确保 I_{MP}可行。此外，对效用补偿$\sum_{j=1}^{n} \delta_j \geq 0$ 的限制给出:

$$\sum_{j=1}^{n} \big(u_j(\sigma) + \beta_j - u_j(\sigma_F)\big) \geq 0 \Leftrightarrow \sum_{j=1}^{n} b_j - \sum_{j=1}^{n} a_j C_j + \sum_{j=1}^{n} \beta_j - \bar{f} \geq 0.$$

由于预算 R 显然已完全耗尽，这等价于 $\sum_{j=1}^{n} b_j - \sum_{j=1}^{n} a_j C_j + \bar{W} + \bar{f} - \sum_{j=1}^{n} b_j - \bar{f} \geq 0$，因此我们有 $\sum_{j=1}^{n} a_j C_j = \sum_{j=1}^{n} w_j C_j \leq \bar{W}$。由此，$I_{MP}$是一个 YES-实例。两个问题中不等式的等价性完成了该归约。□

请注意，如果初始调度为 σ_{SYS} 而非 σ_F，同样的论证也成立。事实上，具体的初始调度与归约无关；重要的只有其总效用值 $\sum_j u_j(\sigma_F)$。因此，我们可以断言，即使初始解是系统最优调度 σ_{SYS} 且效用函数为线性函数， $1|resch\ \sum \delta_j \geq 0|u_{n+1}$ 仍然是弱 NP 困难的。

现在我们来讨论单代理问题的三个版本中的最后一个版本，即1|重新调度 $u_j \geq u_F|u_{n+1}$，并证明该问题可以高效求解。

定理 16 问题（14） 1|重新调度 $u_j \geq u_F|u_{n+1}$ 可以在 $O(n^2 logP)$ 时间内求解。

对于问题（14），从初始系统最优解 σ_{SYS} 出发，并因此将下界 u_F 替换为 u_{SYS}，不会改变定理 16 的证明中所述过程的有效性；该过程适用于个体效用值的任意下界。因此，我们可以得出结论： 1|重新调度 $u_j \geq u_{SYS}|u_{n+1}$，即具有初始解 σ_{SYS} 和线性效用函数的问题（14），是弱 NP 困难的。

我们以对上述结果的几点补充观察结束本节 t.

- 如果不打算向代理 1、.....、 n提供补偿，即 $R = 0$，则问题（14）可以通过一个简化的过程求解（同样只需 $O(n^2)\log(\sum_{j=1}^{n+1} p_j)$时间）。与定理 14 的证明一样，我们对作业 $n+1$ 的完工时间 C_{n+1} 进行二分搜索。对于二分搜索中的每个目标值 C^T，我们得到作业$n+1$ 的相应交期 d_{n+1}^T （其他所有作业均没有交期）。因此，根据定理 11，每次迭代中产生的判定问题都可以在 $O(n^2)$时间内求解。如果所得调度的最大最小效用值小于 u_F，我们就需要考虑更大的目标值；否则，可以测试更小的目标值。

- 显然，定理 16 证明中描述的过程也可以改写为以下情形：我们希望最大化原始作业 1、.....、 n中的某一个作业 i 的效用。在这种情况下，二分搜索的范围被限制在由 p_i 和 MaxMinGreedy确定的完工时间$C_i(\sigma_F)$之间。

- 也可以扩展该过程，考虑多个应尽可能获得高效用（因此完工时间最短）的作业。例如，我们可以按字典序最大化按序排列的作业列表 j_1、.....、 j_k $k \leq n$的效用。通过上文概述的二分搜索过程确定列表中第一个作业 j_1 的最早完工时间 $C_{j_1}^*$ 后，我们将该值设为交期 $d_{j_1} := C_{j_1}^*$，然后继续迭代，最小化列表中下一个作业的完工时间。由此得到的判定问题仍可通过定理 11 求解。

7 在双层设置中强制执行目标序列

与领导者-跟随者场景类似，外部方（领导者）可能希望强制采用某种调度 σ_T，例如，通过最小化完工时间之和来优化吞吐量，或最大化总效用。在这种双层设置中，跟随者仍会求解问题 $1|u_j = b_j - a_j C_j|u_{\min}$ ，为所有作业找到一个公平解 σ_F ；但领导者会通过修改各作业的效用函数来影响这一过程的结果。

首先，类比第 5.2 节，假设领导者可以选择通过增加 Δb_j 来提高截距值 b_j，或通过从a_j中减去 Δa_j来降低斜率。因此，要求 $\Delta b_j \geq 0$，分别为 $0 \leq \Delta a_j \leq a_j$。这样，领导者便能影响跟随者根据修改后的效用函数确定的序列。我们的问题是，求使跟随者生成的公平解 σ_F（即最小效用最大化）对应于所需目标序列 σ_T的最小 $\sum_j \Delta b_j$，分别为 $\sum_j \Delta a_j$。这种设置不足以构成有意义的问题定义，因为最小效用最大化会留下大量最小效用相同的不同序列。对于所有公平性并列的解，领导者无法影响跟随者选择哪个序列。因此，我们将更明确地规定跟随者采取的行动，并假设它使用 MaxMinGreedy 来确定公平解。

根据上文介绍的 Graham 类记法，我们分别将上述两个问题记为$\tilde{}$ $\tilde{}$ $1|\sigma_F = \sigma_T, u_j = b_j - a_j C_j|\sum b_j$和 $1|\sigma_F = \sigma_T, \ u_j = b_j - \tilde{a}_j C_j|\sum \tilde{a}_j$。

这些问题可以按如下方式求解。按照目标序列 σ_T给出的完工时间递增顺序，为作业编号，即:

$$C_j(\sigma_T) \geq C_{j-1}(\sigma_T), \quad j = 2, \ldots, n.$$

核心思路是，对于 MaxMinGreedy 在时间 C_j调度作业 $j \geq 2$，只需选择参数 Δb_j或 Δa_j，使得对所有 $\ell = 1$，.....，$j-1$，均有u_j（C_j）$\geq u_\ell$（C_j）。完工时间较早的所有作业 ℓ，其数值此后都不会再改变。相反，完工时间晚于 C_j 的所有作业，在确定其数值时都会考虑新的效用值 j。在任意时刻 T， MaxMinGreedy 只考虑要安排在 T之前的作业。因此，在选择 Δb_j（分别为 Δa_j）时，无须考虑序列中靠后的作业。

形式上，我们首先选择 $\Delta b_1 := 0$（分别为 $\Delta a_1 = 0$），然后对于 $j \geq 2$，通过设置 $\Delta b_j := \max\{0, b_\ell - a_\ell C_j - b_j + a_j C_j, \ell = 1, \ldots, j-1\}$来增大 $b_j\tilde{}$ 值。注意，通过增大 b_j 值，可以施加任意所需的目标序列。

要减小 a_j，我们设置 $\Delta a_j := \max\{a_j - \tilde{a}_\ell + \frac{b_\ell - b_j}{C_j}, \ell = 1, \ldots, j-1\}$。如果这一选择得到 $\Delta a_j > a_j$，就会产生递增的效用函数，从而违反非递增效用这一一般假设。因此，并非所有所需的目标序列都能通过减小 a_j 值来实现（不过，若也允许增大 a_j，情况就会有所不同，见下文）。

如果跟随者旨在最大化全局效用，就会按照 a_j/p_j 非递增的顺序调度作业。显然，在这种情况下，修改截距 b_j 毫无用处。至于斜率 a_j，则应进行调整，使这种排序对应于领导者所期望的调度 σ_T。

在所得问题 $1|\sigma_{SYS} = \sigma_T, \ u_j = b_j - \tilde{a}_j C_j|\sum \Delta a_j$中，我们必须选择 Δa_j，使得对所有 $\ell < j$均有 $\tilde{a}_j/p_j \leq \tilde{a}_\ell/p_\ell$。由此得到

$$\Delta a_j := \max\{0, a_j - p_j \cdot \min_{1 \leq \ell < j} \{\frac{\tilde{a}_\ell}{p_\ell}\}\}.$$

我们还可以扩展模型，允许在两个方向上修改 b_j，也就是说

Δb_j 可以为正或负，此时目标函数要求所有 Δb_j的绝对值之和最小化。请注意，在这种情况下，从全局角度来看，目前考虑的作业 j的截距增大是否有利，还是降低其他所有作业的截距更有利，并不明确；而这些截距迄今阻碍了 j 被MaxMinGreedy选中。降低其他作业的截距或许有利于算法后续的决策。对于斜率，也可以考虑类似的扩展，即允许 Δa_j为负数，也就是增大负斜率。显然，对于这一一般情形，上述方法不再适用，而且我们尚不知道如何为由此产生的优化问题设计组合算法。不过，我们可以给出一个线性规划表述，这表明该问题仍然是多项式时间可解的。为方便起见，我们用 $P_j := p_1 + \ldots + p_j$ 表示作业 j所需的完工时间。

$$\text{最小化} \sum_{j \in N} |\Delta b_j| \tag{15}$$

$$\text{s.t. } \Delta b_j - \Delta b_i \geq b_i - b_j - (a_i - a_j)P_j \quad 1 \leq i < j \leq n \tag{16}$$

$$\Delta b_j \in \mathbb{R} \qquad j = 1, \ldots, n \tag{17}$$

条件 (16) 由作业 j 在时间 P_j 的 MaxMinGreedy 选择准则推导而来，即 $\Delta b_j + u_j(P_j) \geq \Delta b_i + u_i(P_j)$。目标函数中的绝对值可以通过线性规划建模的标准方法处理。

修改斜率时也适用类似的模型。

$$\text{最小化} \sum_{j \in N} |\Delta a_j| \tag{18}$$

$$\text{s.t. } (\Delta a_j - \Delta a_i)P_j \geq b_i - b_j - (a_i - a_j)P_j \quad 1 \leq i < j \leq n \tag{19}$$

$$\Delta a_j \leq a_j, \quad \Delta a_j \in \mathbb{R} \quad j = 1, \ldots, n \tag{20}$$

请注意，与情形 $\Delta a_j \geq 0$ 相反，在一般情况下，通过适当选择 Δa_j，可以实现任意目标序列。

对于对应的情形 $1|\sigma_T := \sigma_{SYS},\ u_j = b_j - \tilde{a}_j C_j|\sum \Delta a_j$，其中 Δa_j 可能为负，我们将 (19) 替换为

$$\Delta a_j - \frac{p_j}{p_i}\Delta a_i \leq a_j - \frac{p_j}{p_i}a_i \quad 1 \leq i < j \leq n. \tag{21}$$

对于这种情况，我们还可以基于动态规划推导出一种简单的组合算法。该算法基于以下观察：当我们选择 Δa_j，使作业按 $\tilde{a}_j/p_j$的非递增顺序排列时，新斜率的设定方式总能使新的比值 $\tilde{a}_j/p_j$ 取到 n 原始值 a_i/p_i之一。令 $\mathcal{R} := \{a_i/p_i \mid 1 \leq i \leq n\}$为所有相关比值的集合。否则，令 J_r 为满足 $\tilde{a}_j/p_j = r$和 $r \notin \mathcal{R}$的所有作业 j 的集合。将 r 边际增加至所有作业 J_r $r + \epsilon$，可以使正 Δa_j 作业的斜率缩减减少ϵp_j（例如，这些作业为 J_r^+），但要求所有负 Δa_j 进一步减少ϵp_j（对于作业 J_r^-）。对于将 r 边际减少至 $r - \epsilon$，交换正负号后也同样适用。因此，目标函数（18）的总增量

为 $\sum_{j\in J_r^-}\epsilon p_j-\sum_{j\in J_r^+}\epsilon p_j$（以达到 $r+\epsilon$），而以达到$r-\epsilon$时则为 $-\sum_{j\in J_r^-}\epsilon p_j+\sum_{j\in J_r^+}\epsilon p_j$。显然，这两项贡献中必有一项为负，因此会改善目标函数。沿此方向，ϵ 的值可以继续扩展，直至达到 R 中的最近值。

我们引入一个动态规划数组，用于表示将作业j 的比率设为 $r\in\mathcal{R}$的成本。形式化地，我们定义：

$c_j[r]=$ 将 $\tilde{a}_j/p_j=r$ 设定为某个值的成本，加上使所有作业的比率 $\tilde{a}_i/p_i$ $i=j+1$、.....、n 按非递增顺序排序所需的最小成本。

$c_j[r]$ 的计算按 j递减的顺序进行，也就是说，对于每个$r\in\mathcal{R}$，依次取 $j=n$、$n-1$、.....、1。要得到 $\tilde{a}_j/p_j=(a_j-\Delta a_j)/p_j=r$，我们需要考虑 $\Delta a_j=a_j-p_j\cdot r$的成本。再加上作业 $j+1$、.....、n按正确顺序排列的成本（该成本已计入 $c_{j+1}[r]$），可得

$$c_j[r]:=|a_j-p_j\cdot r|+\min_{\rho\in\mathcal{R}}\{c_{j+1}[\rho]:\rho\le r\}. \tag{22}$$

按递增顺序遍历 r 的取值，我们可以在线性时间内计算 $c_j[r]$，从而使求解 $1|\sigma_T:=\sigma_{SYS}$、$u_j=b_j-\tilde{a}_jC_j|\sum\Delta a_j$（其中 Δa_j 任意）的总运行时间为 $O(n^2)$。

8 结论

在这项工作中，我们研究了广泛的一类多智能体调度问题，旨在为效用函数、时间约束以及调整自身参数的灵活程度各不相同的代理找到公平解（按极大极小意义）。我们的分析阐明了释放日期、交期和代理特定的效用模型如何影响问题的内在难度，以及如何利用对输入参数的受控修改来获得理想的调度方案。基于这些见解，我们开发了能够计算公平调度的精确方法；此外，每当对所研究的问题设定而言有意义时，我们也研究了效率版本，即最大化效用总和。我们还探讨了可调整的效用函数和领导者—跟随者情景，揭示了预算受限的效用修改等外部干预如何影响最终的调度方案，并展现了多代理环境中公平性、激励与策略控制之间的联系。

值得一提的是，我们的研究可扩展至更一般的公平性概念，例如卡莱—斯莫罗丁斯基公平解，即最大化各代理之间的归一化效用最小值（Kalai and Smorodinsky，1975年）。更确切地说，考察任意半主动调度方案中代理 j可能获得的效用值范围$I_j=\max_{\sigma\in\mathcal{S}}\{u_j(\sigma)\}-\min_{\sigma\in\mathcal{S}}\{u_j(\sigma)\}$，代理 j 的归一化效用 $\bar{u}_j$ 定义为比率 $\bar{u}_j(\sigma)=u_j/I_j$. 例如，回顾定义2，对于线性效用函数且没有释放日期的情形，如果$P=\sum_j p_j$，那么 $I_j=a_j(P-p_j)$，因此对于任意给定的调度方案，所有 j 的归一化效用均已知。如果能够预先计算出每个代理的这一范围，那么任何能够计算 u_F 的算法也可用于求得卡莱—斯莫罗丁斯基公平解（尽管最终得到的调度方案可能不同）。

未来的研究可以考虑 m 并行机器的情形，这些机器的规格可能各不相同。考察 $m = 2$ 相同并行机器这一特殊情形，可以很容易证明，即使所有作业都具有相同的线性效用函数，我们的问题仍然是弱 NP 困难。与第4节.3的思路类似，可以针对后一问题设计一种基于动态规划的伪多项式算法。

参考文献

Agnetis, A., Benini, M., Nicosia, G. 和 Pacifici, A.（2025年）。双代理单机调度中效用与公平性之间的权衡。《欧洲运筹学杂志》，323(3):767–779。Agnetis, A., Chen, B., Nicosia, G. 和 Pacifici, A.（2019年）。双代理单机调度问题中的公平代价。《欧洲运筹学杂志》，276(1):79–87。Agnetis, A., Mirchandani, P. B., Pacciarelli, D. 和 Pacifici, A.（2004年）。双竞争代理调度问题。《运筹学》，52(2):229–242。Agnetis, A., Nicosia, G., Pacifici, A. 和 Pferschy, U.（2015年）。采用中央选择机制的双代理任务链调度。《调度学报》，18(3):243–261。Agnetis, A., Pacciarelli, D. 和 Pacifici, A.（2007年）。多代理单机调度。《运筹学年鉴》，150(1):3–15。Baker, K. R. 和 Smith, J. C.（2003年）。机器调度的多准则模型。《调度学报》，6(1):7–16。Baptiste, P.（1999年）。在处理时间相等的单机上最小化迟到作业加权数的多项式时间算法。《调度学报》，2(6):245–252。Baptiste, P.（2000）。在相同并行机上调度等长度作业。《离散应用数学》，103(1):21–32。Bertsimas, D., Farias, V. 和 Trichakis, N.（2011年）。公平的代价。《运筹学》，59(1):17–31。Biskup, D.（2008年）。学习效应调度研究现状综述。《欧洲运筹学杂志》，188(2):315–329。Burkard, R., Dell'Amico, M. 和 Martello, S.（2012年）。指派问题。工业与应用数学学会。Carlier, J.（1981年）。单机问题及多项式算法。*QUESTIO*，5(4):219–228。Chen, B., Potts, C. N. 和 Woeginger, G. J.（1998 年）。机器调度综述：复杂性、算法与可近似性。载于 Du, D.-Z. 和 Pardalos, P. M. 编，《组合优化手册：第1–3卷》，第1493–1641页。施普林格美国，波士顿，马萨诸塞州。

Chen, R., Li, S.-S., Yuan, J. 和 Zhao, Q.（2026年）。在单机或均匀并行机上调度等长度作业的竞争性多智能体调度。《欧洲运筹学杂志》。即将出版。Cheng, T. E., Kovalyov, M. Y. 和 Shakhlevich, N. V.（2006年）。具有可控释放日期和处理时间的调度：总完工时间最小化。《欧洲运筹学杂志》，175(2):769–781。Correa, J. R. 和 Queyranne, M.（2012年）。以总加权完工时间为社会成本时，受限均匀机调度的均衡效率。《海军研究后勤学》，59(5):384–395。Curiel, I., Pederzoli, G. 和 Tijs, S.（1989年）。排序博弈。《欧洲运筹学杂志》，40(3):344–351。Dedehayir, O. 和 Steinert, M.（2016年）。技术炒作周期模型：综述与未来方向。《技术预测与社会变迁》，108:28–41。Gawiejnowicz, S.（2008年）。时间相关调度。《理论计算机科学专著》。EATCS丛书。施普林格。Gordon, V. S., Proth, J.-M. 和 Chu, C.（2002年）。交货期指派与调度：SLK、TWK及其他交货期指派模型。《生产计划与控制》，13(2):117–132。Hall, N. G. 和 Potts, C. N.（2004年）。针对新订单的重新调度。《运筹学》，52(3):440–453。Hermelin, D., Molter, H., Niedermeier, R., Pinedo 和 Shabtay, D.（2025年）。重复性调度中的公平性。《欧洲运筹学杂志》，323(3):724–738。Hoeksma, R. 和 Uetz, M.（2019年）。相关机上功利主义调度博弈的无政府状态代价。《离散优化》，31:29–39。Höhn, W. 和 Jacobs, T.（2015年）。非线性成本单机调度中Smith规则的性能。《*ACM*算法汇刊》，11(4)。Huang, X. 和 Wang, M.（2014年）。具有时间和位置相关处理时间的单机分组调度。《优化快报》，8:1475–1485。Kalai, E. 和 Smorodinsky, M.（1975年）。Nash谈判问题的其他解。《计量经济学》，43(3):513–518。Kämpke, T. 和 Radermacher, F. J.（2015年）。分布函数的广义逆。《经济学与数学系统讲义》，679:9–28。Kellerer, H., Pferschy, U. 和 Pisinger, D.（2004年）。背包问题。施普林格。

Lawler, E. (1978). 在满足优先约束的条件下，对作业排序以最小化总加权完工时间。《离散数学年刊》，2:75–90。Lawler, E. L. (1973). 满足优先约束的单机最优排序。《管理科学》，19(5):544–546。Lenstra, J., Rinnooy Kan, A., and Brucker, P. (1977). 机器调度问题的复杂性。载于 Hammer, P., Johnson, E., Korte, B., and Nemhauser, G. 编，《整数规划研究》，第1卷《离散数学年刊》，第343–362页。爱思唯尔。Lenstra, J. K. and Shmoys, D. B. (2020). 调度学要素。《CoRR》，abs/2001.06005。Li, S.-S. and Yuan, J.-J. (2020). 多代理单机调度问题：最小化总加权延误工作量。《调度学报》，23:497–512。Lu, Y.-Y., Wang, J.-J., and Huang, X. (2015). 基于位置和处理时间总和的处理时间的作业调度。《应用数学建模》，39(14):4013–4021。Moore, J. M. (1968). 一项用于最小化延迟作业数量的单机作业排序算法。《管理科学》，15(1):102–109。Nicosia, G., Pacifici, A., and Pferschy, U. (2017). 分配有界资源的公平代价。《欧洲运筹学杂志》，257(3):933–943。Nicosia, G., Pacifici, A., Pferschy, U., Resch, J., and Righini, G. (2021). 使用后进先出缓冲区对作业进行最优重新调度。《调度学报》，24(6):663–680。Niu, A., Totschnig, A., and Vetta, A. (2023). 公平算法设计：公平且高效的机器调度。载于 Deligkas, A. and Filos-Ratsikas, A. 编，《算法博弈论》，第239–256页。施普林格。Raut, S., Gupta, J. N. D., and Swami, S. (2008a). 作业价值随时间递减的单机调度。《运筹学会杂志》，59(1):105–118。Raut, S., Swami, S., and Gupta, J. N. (2008b). 作业价值随时间递减的有容量限制单机调度。《国际生产经济学杂志》，114(2):769–780。Rener, E., Salassa, F., and T’kindt, V. (2025). 针对新订单、满足最大和总时间扰动约束的单机重新调度。《调度学报》，28:469–482。Shabtay, D. and Steiner, G. (2007). 可控加工时间调度综述。《离散应用数学》，155(13):1643–1666。Simons, B. (1978). 单处理器调度的快速算法。载于《第19届计算机科学基础年度研讨会》，第246–252页。Smith, W. E. (1956). 单阶段生产的多种优化方法。《海军研究后勤季刊》，3(1–2):59–66。

Strusevich, V. A. and Rustogi, K. (2016). 具有灵活维护的调度。载于《具有时变效应和速率调节活动的调度》，国际运筹学与管理科学丛书第243卷，第291–315页。施普林格。
Townsend, W. (1978). 完工时间二次惩罚函数下的单机问题：一种分支定界解法。《管理科学》，24(5):530–534。Vardi, S. and Haskell, W. (2025). 调度稀缺资源的公平代价。《运筹学》，73(6):3104–3117。Wang, D., Yu, Y., Yin, Y., and Cheng, T. C. E. (2021). 多任务处理下的多智能体调度问题。《国际生产研究杂志》，59(12):3633–3663。Wellman, M. P., Walsh, W. E., Wurman, P. R., and MacKie-Mason, J. K. (2001). 分散式调度的拍卖协议。《博弈与经济行为》，35(1):271–303。Yuan, J. (2017). 固定数量竞争智能体下的单机多代理调度：最小化延期作业数量与最大完工时间的加权和。《组合优化杂志》，34(2):433–440。Zhang, Y., Zhang, Z., and Liu, Z. (2020). 最小化总完工时间的双智能体调度博弈的公平代价。《组合优化杂志》，44:2104–2122。

附录

定理2的证明

如果对于所有 $j \in J$, u_j， C_j 都是作业j的完工时间的任意非递增函数，那么 MaxMinGreedy 会在 O（n^2）时间内计算出 $1|u_j|u_{\min}$ 的一个最优解。

对于给定的调度 σ，令 $z(\sigma) = \min^{j\in J} u_j(C_j)$表示其对应的解值。令$\sigma_F$ 为一个最优调度，其中 $i \in J$ 是最后一个作业：由于不失一般性，我们可以假设 σ_F没有空闲时间，因此 i 在 $C_i(\sigma_F) = P = \sum_{j\in J} p_j$时完工。

令 $i \in J$为算法选出的作业，即 $i = \arg\max_{h\in J}\{u_h(P)\}$，并通过反证法假设 $u_i(P) < u_i(P)$。考虑从 σ_F 得到的调度 $\tilde{\sigma}$：只需将作业 i 推迟，并将其排在最后一个，紧接在 i之后。在 $\tilde{\sigma}$ 中，除 i外的所有作业的完工时间均未增加，因此，根据我们的目标函数定义：对于所有 $h \in J \setminus \{i\}$，$z(\sigma_F) \le u_h(C_h(\sigma_F) \le u_h(C_h(\tilde{\sigma}))$。另一方面， $z(\sigma_F) \le u_i(C_i(\sigma_F)) = u_i(P) < u_i(P) = u_i(C_i(\tilde{\sigma}))$。

总之， $z(\sigma_F) \le \min_{h\in J}\{u_h(C_h(\tilde{\sigma}))\} = z(\tilde{\sigma})$，因此 $\tilde{\sigma}$ 也同样是最优的。可以在时间 $P - p_i$ 处递归地重复这一论证，从而证明 MaxMinGreedy的正确性。□

定理 3 的证明

问题 $1|r_j, u_j|u_{\min}$ 是强 *NP* 困难的，即使在线性效用函数下也是如此。

考虑一个带有交期和释放日期、以最小化最大延迟 $L_{\max}$ 为目标的调度问题实例 I_L ，即 $1|r_j|L_{\max}$。Lenstra 等人（1977 年）已证明该问题是强 NP 困难的。我们可以构造一个具有相同释放日期和效用函数 $u_j = d_j - C_j + M$的问题实例，其中 $M > 0$ 足够大，以保证对于任何合理的 C_j取值，$u_j \ge 0$ 。将我们问题的解视为 I_L的解，可以很容易地看出，延迟 $L_j := C_j - d_j = -u_j + M$。如果存在一个满足$u_F \ge K + M$的解，那么对于 I_L，就存在一个满足 $L_{\max} \le -K$ 的解，反之亦然。□

定理 4 的证明

问题 $1|r_j, u_j|u_{\min}$ 是弱 *NP* 困难的，即使效用函数是线性的且只有一个释放日期非零。

对于仅有一个释放日期的情况， $r_{n+1} \ne 0$, $1|r_j, u_j|u_{\min}$ 可以从 Partition归约得到。令 I_P 为 Partition 的一个实例，其由 n 个整数 w_1、.....、w_n 定义，并令 $B = \frac{1}{2}\sum_{i=1}^{n} w_i$。该问题要求找出一个数字子集 S，使其总和等于 B。我们调度问题对应的实例 I 如下。共有 $n + 1$ 个作业，其中前 n 个作业对应于 I_P中的各个项目。数据如下：

$$p_j = \begin{cases} w_j & \text{for } j = 1, 2, \ldots, n \\ B & \text{for } j = n + 1 \end{cases}$$

$$r_j = \begin{cases} 0 & \text{for } j = 1, 2, \ldots, n \\ B & \text{for } j = n+1 \end{cases}$$

$$u_j(C_j) = \begin{cases} 2B - \frac{1}{3}C_j & \text{for } j = 1, 2, \ldots, n \\ 5B - 2C_j & j = n+1 \end{cases}$$

如果 I_P 是一个 YES-实例，那么在 I 中，可以将 S 中的作业安排在时间 0 到 B之间，然后在时间 B开始作业 $n+1$，并将剩余作业 $\{1, \ldots, n\} \setminus S$安排在时间 $2B$到$3B$之间。这样，作业 $n+1$ 的效用为 $5B - 2 \cdot 2B = B$，而在 1、.....、n中最后调度的作业的效用为 $2B - \frac{1}{3}3B = B$，即达到最低效用的作业，从而得到 $u_F = B$。

如果 I_P是一个 NO-实例，可以选择在时间 B调度作业 $n+1$，再次获得效用 B。由于在时间 B 之前调度的作业总长度必须严格小于 B（NO-实例），因此从时间 $2B$ 开始、在作业 $n+1$ 之后调度的剩余作业总长度至少为$B+1$。因此，最后一个作业的效用至多为 $2B - \frac{1}{3}$（$2B + B + 1$）$\leq B - \frac{1}{3}$。否则，如果作业 $n+1$ 的调度时间晚于 B，其效用至多为 $5B - 2$（$B + 1 + B$）$= B - 2$。在这两种情况下，都有 $u_F < B$，这意味着 I 的最优解将决定 PARTITION。 □

定理 16 的证明

问题 (14) $1|resch\ u_j \geq u_F|u_{n+1}$ 可以在 $O(n^2 logP)$ 时间内求解。

问题 $1|resch\ u_j \geq u_F|u_{n+1}$ 可以通过结合本文的两个结果（即定理 1 和第 5.1 节给出的过程）来求解。首先，我们对作业 $n+1$ 的最小完工时间 C_{n+1} 进行二分搜索，迭代 $O(\log\ P)$ 次。显然，C_{n+1}必须介于 p_{n+1} 和 $P = \sum_{j=1}^{n+1} p_j$之间。二分搜索的每个目标值 C_{n+1}^T 都会为 $C \leq C_{n+1}^T$ 指定一个效用函数 $u_{n+1}(C) = M$，而在其他情况下则指定 $u_{n+1}(C) = 0$。随后，我们可以运行MAXMINGREEDY，在$O(n^2)$ 时间内获得一个可行调度 σ^T，其最大最小效用为 $u_{\min}(\sigma^T)$。对于该调度，我们采用第 5.1 节给出的线性时间过程，通过常数平移来提高效用函数 Δb_j，这些平移代表支付 β_j，且给定预算上界 $B = R$（该方法也适用于任意效用函数）。请注意，在此提高过程中，作业 $n+1$ 的（辅助）效用函数 $u_{n+1}(C)$ 始终不会被修改。如果改进后的最大最小值至少为 u_F，我们就可以继续二分搜索并选取大于 C^T的目标值；否则，选择较小的目标值。□