

## RESEARCH ARTICLE

# A Lower Bound for Minimizing Waiting Time in Coexisting Virtual and Physical Worlds

YU-CHUAN CHEN<sup>1</sup> AND JEN-YA WANG<sup>2</sup> <sup>1</sup>Department of Intelligent Technology and Application, Hungkuang University, Taichung 433, Taiwan<sup>2</sup>Department of Information Management, National Taichung University of Science and Technology, Taichung 404, Taiwan

Corresponding author: Jen-Ya Wang (jywang@nutc.edu.tw)


This work was supported by the National Science and Technology Council under Grant MOST 111-2410-H-241-001 and Grant NSTC 112-2410-H-025-045.

**ABSTRACT** To balance customer satisfaction across virtual and real-world interactions, we focus on enhancing service for dine-in customers at restaurants that typically prioritize online orders, such as those on Uber Eats. Utilizing three-agent scheduling strategies that adhere to each agent's specific requirements—whether they are hard constraints or soft objectives—we effectively manage various types of orders, including immediate individual online orders, group reservations, and oral requests from dine-in customers. This approach significantly reduces waiting times and improves overall customer satisfaction. We propose a branch-and-bound algorithm with a tight lower bound based on preemption, which prioritizes agents A and B while reducing the total waiting time for agent C, representing dine-in customers. Computational experiments reveal that our algorithm significantly reduces total waiting times compared to existing two-agent scheduling strategies, demonstrating its effectiveness. Despite its efficiency, the algorithm incurs computational overheads, particularly with larger problem sizes. Our unique lower bound can be extended to other industries requiring multiple constraints or objectives. For example, in the film and television industry, real actors (represented by agent A) need to align their shooting times with 3D studios (represented by agent B) and stunt doubles for virtual avatars (represented by agent C) for scenes where they interact. That is, more agents are required to accommodate their constraints and objectives in such a scenario.

**INDEX TERMS** Multi-agent scheduling, branch-and-bound algorithm, lower bound, waiting time, completion time.

## I. INTRODUCTION

In today's technologically advanced world, the integration of virtual and real environments in applications is increasingly common, which underscores the importance of effectively allocating resources between these two worlds. As technologies such as VR, MR, and AR become essential in areas like medical training, education, and surgical guidance, they highlight the critical need to manage resources across both virtual and real settings [1], [2]. For example, Hsiao et al. [2] observed that the application of AR is based in the real world, as the Pokémon game requires access to the participants' location information in the real world and also requires participants to pay, yet many players are tirelessly enthusiastic.

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng .

This growing interdependence between virtual and physical realms demands sophisticated resource management strategies that can accommodate the unique needs of each, without compromising the efficiency and quality of user experiences.

Time, as a crucial resource in both the virtual and real worlds, requires careful management to maximize efficiency and meet the expectations of all users. The challenge lies not only in effectively allocating time but also in ensuring that this allocation benefits users in both realms equally. Effective time management strategies are vital for maintaining competitive operations and ensuring customer satisfaction across both platforms. For example, Su and Wang [3] indicated that in the gaming industry, the cost of a game project often exceeds tens of millions of dollars and requires multiple professionals in the real world (e.g., real desert oasis scenery) or the virtual world (e.g., virtual 3D paradise scene). A delay

in any job could lead to a daily penalty of \$100. Therefore, it is crucial to schedule all jobs without tardiness. By optimizing how time is allocated and used, businesses can better synchronize their services to enhance overall user engagement and satisfaction.

The emphasis on online ratings often leads to a resource allocation bias towards virtual services, adversely affecting real-world interactions. This trend is particularly evident in sectors like dining, where restaurants might prioritize online orders over dine-in customers due to the perceived importance of online reviews, e.g., [4]. Such practices can diminish the quality of service for dine-in customers and lead to a disparity in customer satisfaction. Addressing this issue requires a balanced approach that considers both virtual and real-world demands, ensuring that no customer experience is undervalued.

In view of the above observations, we aim to balance resource sharing in the virtual and real worlds. Due to limited resources, it is challenging to ensure equitable resource distribution between these two worlds. Decision makers will often make choices that align with their corporate culture, business rules, and legal standards to best serve their preferred customers. While it is understood that not all customers can be prioritized equally and resources are finite, we must strive to prevent any undue disadvantage. Thus, to fairly accommodate the interests of customers in both worlds, it is essential to develop a new job scheduling algorithm.

This study introduces a novel three-agent scheduling problem to address these imbalances. The proposed branch-and-bound algorithm, with a tight lower bound, is designed to optimize customer satisfaction by effectively balancing the needs of both virtual and real-world customers. By prioritizing the scheduling needs of various agents, this algorithm manages the competing demands of maintaining high online ratings and ensuring minimal wait times for dine-in customers. The model's effectiveness has been validated through computational experiments, demonstrating its capability to significantly reduce wait times without compromising service quality in either realm. Consequently, this model can be extended to other industries that require addressing multiple constraints or objectives, such as the film and television industry, where real actors, virtual avatars, and expensive 3D studios each have unique requirements.

## II. RELATED WORK

In this section, we will clearly identify the gaps between previous research and this study by exploring waiting time management in both virtual and real environments, as well as multi-agent scheduling and lower bound design.

As the integration of virtual and real worlds becomes more prevalent, competition for critical resources, such as time, inevitably intensifies. For instance, [1] illustrated how VR is employed to enhance life skills for autistic children by developing a serious game specifically designed to train them to cross streets safely. In such scenarios, it is crucial that the computation determining whether a real person collides

with a VR vehicle is not excessively delayed, as prolonged retrieval of information from the virtual world could result in a loss of realism. Similarly, in the case of Pokémon, when a player captures a monster, the response must be quick enough to ensure it is recognized by other AR players and prevent the peculiar situation where the same virtual monster is claimed by multiple players. Inadequate management of these resources could undermine the essential purpose of integrating the virtual and real worlds.

In our research, waiting time serves as a crucial indicator of customer satisfaction, representing the average duration each customer spends within a system from the moment they enter a store or when a service commences. This includes both idle and processing times, with shorter durations indicating better customer satisfaction [5]. In the field of job scheduling, average waiting time is calculated as  $\sum_{i=1}^n W_j(\pi)/n$ , akin to the total completion time,  $\sum_{i=1}^n C_j(\pi)$ , where  $\pi$  denotes the job scheduling scheme. Mathematically, the lowest average waiting time and the lowest total completion time are considered equivalent indicators. Previous studies, such as [6] and [7], have primarily focused on minimizing total completion time for all jobs. From a producer's perspective, a reduced total completion time indicates more efficient scheduling; from a customer's perspective, shorter average waiting times contribute to higher satisfaction. Thus, from both business and customer viewpoints, managing waiting time remains a pivotal area of research.

Inspired by two-agent scheduling, this study adopts three-agent scheduling strategies to model the allocation of time resources between the virtual and real worlds. The concept was initially introduced in [8] and [9], illustrating scenarios where a company might need to share machine resources between two departments due to limited availability, necessitating appropriate scheduling to satisfy both departments' demands. This involves distinguishing between soft objectives and hard constraints. However, previous research predominantly focused on two-agent scheduling [10], [11], [12], [13]. In today's integrated virtual and real world, where the number of stakeholders continues to grow and each has distinct positions, traditional two-agent scheduling may no longer adequately represent the vast array of objectives and constraints. Taking the film industry as an example, the planning department should try to schedule top-paid actors to shoot for several consecutive days at the end of December to reduce idle costs. However, the technical department is engaged in filming elsewhere during December, and the human resources department opposes having employees work over the holidays. Such demands from these three different perspectives can lead to conflicts in time management. This indicates a need for scheduling models that incorporate more agents.

TABLE 1 shows research related to multi-agent scheduling focusing on completion time and waiting time since 2021. The data indicates that the majority of researchers have concentrated on improving completion time in two-agent scheduling scenarios, with only a few studies addressing

TABLE 1. Recent literature on two-agent and three-agent scheduling.

Objective	Agents	Two agents	Three agents
Waiting Time		[14]	
Completion Time		[13-21]	[22]

three-agent scenarios. Therefore, the scheduling algorithms from these studies are often not directly applicable to our problem involving three agents. Moreover, most research prioritizes total completion time, considering it from the perspective of decision makers, rather than from the perspective of customer satisfaction, which aims to reduce average waiting times. This highlights the uniqueness of this study, which considers scheduling from the customer’s perspective.

Given the scarcity of algorithms that effectively integrate three-agent scheduling in both virtual and real worlds, we aim to develop a branch-and-bound algorithm to establish a benchmark for future metaheuristic algorithms. It is crucial to note that exact algorithms consistently produce optimal solutions. Although they are time-intensive, their optimal outcomes provide objective benchmarks for accurately evaluating the performance of approximate algorithms. Branch and bound is a well-recognized paradigm within exact algorithms.

The essence of a branch-and-bound algorithm lies in its lower bound. Moreover, while a branch-and-bound algorithm must explore the entire search tree, the computation required is substantial. This necessitates the prevention of futile searches in branches destined to fail. A well-crafted lower bound serves to determine if a branch is certain to fail. Early identification of such branches can significantly conserve computing time. Consequently, a precise lower bound can greatly enhance the efficiency of a branch-and-bound algorithm.

In this study, we aim to develop a lower bound using two of the most classic techniques: preemption and agreeable jobs. Preemption is recognized as a fundamental strategy in lower bound design [23], [24]. In scenarios where a large job cannot be allocated to a machine due to its impending due date or the limited capacity of the machine, a preemptive lower bound will divide the job and assign its surplus part to another machine, ensuring that the lower bound never exceeds the corresponding exact cost. Making jobs agreeable is another straightforward and effective approach for designing lower bounds [25], [26], [27]. To estimate a tight lower bound, researchers will sort the release times, processing times, and due dates in ascending order. This sorting method is simple, time-efficient, and consistently yields a lower bound that does not surpass the actual minimal cost. Therefore, we plan to apply these two techniques to the three-agent scheduling problem in our study to develop a lower bound that closely mirrors the true optimal solution.

In light of the above observations, we need the optimal solutions for resource scheduling in this integrated virtual and real environment. However, due to the complexity of the three-agent scheduling problem in both virtual and real

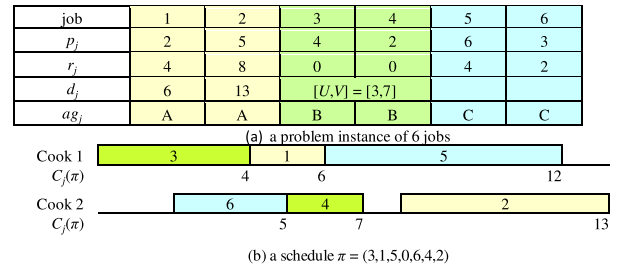


FIGURE 1. A problem instance.

worlds, existing lower bounds are insufficient. Therefore, there is a clear need to develop a new lower bound.

### III. PROBLEM DESCRIPTION

A three-agent scheduling problem for minimizing the total waiting time of real-world customers is formulated as follows. For a given time period, there are  $n$  non-preemptive jobs contracted by three agents, i.e., A, B, and C. Each job  $j$  belongs an agent  $ag_j$  and has a processing time  $p_j$  as well as a release time  $r_j$ , and it is about to be assigned one of  $m$  identical cooks. A cook can process one job at a time. There are two constraints. First, the completion time of each job  $j$  of agent A cannot be greater than its due date, i.e.,  $C_j(\pi) \leq d_j$ , where  $\pi$  means a schedule. That is, agent A has zero tardiness. Second, all the jobs of agent B are released at time 0, i.e.,  $r_j = 0$ ; all their completion times must be within a given due window, i.e.,  $C_j(\pi) \in [U, V]$ . Namely, agent B has zero earliness and tardiness. This hard constraint stems from an observed unfair scenario in the real world where agent B, being dominant, always achieves its objective by undermining the interests of other agents. Moreover, the waiting time of a job  $j$  belonging to agent C is defined as  $W_j(\pi) = C_j(\pi) - r_j$ . Under the above assumptions and constraints, the objective is to minimize the total waiting time for agent C. The objective function is defined as

$$\text{Min } f(\pi) = \sum_{ag_j=C} W_j(\pi), \tag{1}$$

s. t.

$$\begin{aligned} \sum_{ag_j=A} T_j(\pi) &= \sum_{ag_j=A} \max \{0, C_j(\pi) - d_j\} = 0, \tag{2} \\ \sum_{ag_j=B} [E_j(\pi) + T_j(\pi)] \\ &= \sum_{ag_j=B} [\max \{0, U - C_j(\pi)\} + \max \{0, C_j(\pi) - V\}] = 0. \tag{3} \end{aligned}$$

FIGURE 1a illustrates a problem instance and FIGURE 1b depicts a schedule  $\pi = (3, 1, 5, 0, 6, 4, 2)$  for this instance, where the numeral 0 serves as a separator dividing jobs between cooks. Since the jobs of agent B must be completed within [3, 7], jobs 3 and 4 are allocated to cooks 1 and 2 respectively. Furthermore, job 1 is processed by cook 1 at time 4 and completed at time 6, ensuring no tardiness. Similarly, cook 2 begins processing job 2 at time 8, also without tardiness. As a result, agent A has no tardy jobs. Lastly, jobs 5 and 6 are assigned to cooks 1 and 2 respectively,

as early as possible, maximizing efficiency. Consequently, agent C achieves considerable customer satisfaction, with a total waiting time of 11 ( $= 12 - 4 + 5 - 2$ ).

The following features distinguish this problem from previous research: First, there are three agents representing parties in both the virtual and real worlds, making the problem more complex with additional constraints. Second, the jobs associated with each agent have varying due dates or windows, posing challenges for scheduling. Lower bounds that rely solely on simple preemption may compromise the structure of jobs and reduce the effectiveness of these bounds. Third, this study prioritizes two hard constraints—zero tardiness from dominant agents—unlike previous studies where such constraints were often treated as soft objectives. As a result, the development of new exact algorithms and more precise lower bounds is imperative.

#### IV. BRANCH-AND-BOUND ALGORITHM

To optimally solve this problem, we develop an exact algorithm, i.e., a branch-and-bound algorithm, to generate the optimal schedules and ensure solution quality. Initially, we develop some dominance rules to accelerate its execution speed. Then we also propose a lower bound to enhance BB's efficiency. Finally, we introduce the pseudo code of BB step by step.

##### A. DOMINANCE RULES

For convenience, we first introduce some simple notations. Suppose there are two incomplete schedules  $\pi' = (\alpha', \beta)$  and  $\pi = (\alpha, \beta)$ , where  $\alpha'$  and  $\alpha$  are two determined partial schedules and  $\beta$  is the remaining undetermined jobs. The only difference between the two schedules is their two last two jobs in  $\alpha'$  and  $\alpha$ , i.e.,  $\alpha' = (\dots, j, i)$  and  $\alpha = (\dots, i, j)$ . Moreover, the earliest time that job  $i$  in  $\alpha$  or job  $j$  in  $\alpha'$  can start is  $t$ . To show that  $\pi'$  is dominated by  $\pi$ , we must ensure that  $C_j(\pi) \leq C_i(\pi')$ ,  $C_i(\pi) \leq d_i$ , and  $C_j(\pi) \leq d_j$  hold. Since all the dominance rules are similar, we prove the first only.

*Situation 1:* Consider both jobs  $i$  and  $j$  belonging to agent A. In Rules 1–4,  $\alpha$  has an earlier completion time, so  $\pi'$  is dominated by  $\pi$ . In Rules 4–6, although  $\alpha$  is tied with  $\alpha'$  in terms of completion time, we let  $\pi$  dominate  $\pi'$  if  $i < j$ . In this situation, schedules with earlier completion times are deemed winners. In the event of a tie, the job with a smaller job ID wins.

**Rule 1.** If  $ag_i = A, ag_j = A, r_i \leq t < r_j \leq t + p_i$ , and  $t + p_i + p_j \leq d_j$ , then  $\pi'$  is dominated.

*Proof:* We have  $C_j(\pi) = \max\{C_i(\pi), r_j\} + p_j = t + p_i + p_j < r_j + p_j + p_i = \max\{C_j(\pi'), r_i\} + p_i = C_i(\pi')$ . Moreover, we have  $C_i(\pi) = \max\{t, r_i\} + p_i = t + p_i \leq d_j$  and  $C_j(\pi) = \max\{C_i(\pi), r_j\} + p_j = \max\{\max\{t, r_i\} + p_i, r_j\} + p_j = \max\{t + p_i, r_i\} + p_j = t + p_i + p_j \leq d_j$ . The proof is complete. ■

**Rule 2.** If  $ag_i = A, ag_j = A, r_i \leq t$ , and  $t + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 3.** If  $ag_i = A, ag_j = A, t \leq r_i < r_j \leq r_i + p_i$ , and  $r_i + p_i + p_j \leq d_j$ , then  $\pi'$  is dominated.

**Rule 4.** If  $ag_i = A, ag_j = A, t \leq r_i, r_i + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 5.** If  $ag_i = A, ag_j = A, t \leq r_i = r_j, i < j$ , and  $r_i + p_i + p_j \leq d_j$ , let  $\pi'$  be dominated.

**Rule 6.** If  $ag_i = A, ag_j = A, r_i \leq t, r_j \leq t, i < j$ , and  $t + p_i + p_j \leq d_j$ , let  $\pi'$  be dominated.

*Situation 2:* Consider both jobs  $i$  and  $j$  belonging to agent B. In Rules 7–8,  $\alpha$  has an earlier completion time, so  $\pi'$  is dominated. In Rules 9–10, although  $\alpha$  is tied with  $\alpha'$  in terms of completion time, we let  $\pi$  dominate  $\pi'$  if  $i < j$ . Similarly, the situation for agent B is the same as Situation 1 for agent A.

**Rule 7.** If  $ag_i = B, ag_j = B, U - p_i \leq t < U - p_j$ , and  $t + p_i + p_j \leq V$ , then  $\pi'$  is dominated.

**Rule 8.** If  $ag_i = B, ag_j = B, t \leq U - p_i < U - p_j$ , and  $U + p_j \leq V$ , then  $\pi'$  is dominated.

**Rule 9.** If  $ag_i = B, ag_j = B, t \leq U - p_i = U - p_j, i < j$ , and  $U + p_j \leq V$ , let  $\pi'$  be dominated.

**Rule 10.** If  $ag_i = B, ag_j = B, U - p_i \leq t, U - p_j \leq t, i < j$ , and  $t + p_i + p_j \leq V$ , let  $\pi'$  be dominated.

*Situation 3:* Consider both jobs  $i$  and  $j$  belonging to agent C. In Rules 11–14,  $\alpha$  has not only an earlier completion time but also a shorter waiting time, so  $\pi'$  is dominated. In Rules 15–16,  $\alpha$  has a tied completion time but a shorter waiting time, so  $\pi'$  is dominated. In Rules 17–18, although  $\alpha$  is tied with  $\alpha'$  in terms of completion time and waiting time, we let  $\pi$  dominate  $\pi'$  if  $i < j$ . The two jobs belonging to agent C follow the same pattern, comparing completion times first and then job IDs in case of a tie.

**Rule 11.** If  $ag_i = C, ag_j = C, r_i \leq t < r_j \leq t + p_i$ , and  $2(t - r_j) + p_i - p_j < 0$ , then  $\pi'$  is dominated.

**Rule 12.** If  $ag_i = C, ag_j = C, r_i \leq t$ , and  $t + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 13.** If  $ag_i = C, ag_j = C, t \leq r_i < r_j \leq r_i + p_i$ , and  $2(r_i - r_j) + p_i - p_j < 0$ , then  $\pi'$  is dominated.

**Rule 14.** If  $ag_i = C, ag_j = C, t \leq r_i$ , and  $r_i + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 15.** If  $ag_i = C, ag_j = C, r_i \leq t, r_j \leq t$ , and  $p_i < p_j$ , let  $\pi'$  be dominated.

**Rule 16.** If  $ag_i = C, ag_j = C, t \leq r_i = r_j$ , and  $p_i < p_j$ , let  $\pi'$  be dominated.

**Rule 17.** If  $ag_i = C, ag_j = C, t \leq r_i = r_j, i < j$ , and  $p_i < p_j$ , let  $\pi'$  be dominated.

**Rule 18.** If  $ag_i = C, ag_j = C, r_i \leq t, r_j \leq t, i < j$ , and  $p_i = p_j$ , let  $\pi'$  be dominated.

*Situation 4:* Consider job  $i$  belonging to agent A and job  $j$  belonging to agent B. In Rules 19–22,  $\alpha$  has an earlier completion time, so  $\pi'$  is dominated. In Rules 23–24, although  $\alpha$  is tied with  $\alpha'$  in terms of completion time, we let  $\pi$  dominate  $\pi'$  if  $i < j$ . This situation involves comparing the completion times of jobs belonging to different agents A and B.

**Rule 19.** If  $ag_i = A, ag_j = B, r_i \leq t < U - p_j \leq t + p_i$ , and  $t + p_i + p_j \leq V$ , then  $\pi'$  is dominated.

**Rule 20.** If  $ag_i = A, ag_j = B, r_i \leq t$ , and  $t + p_i \leq U - p_j$ , then  $\pi'$  is dominated.

**Rule 21.** If  $ag_i = A, ag_j = B, t \leq r_i < U - p_j \leq r_i + p_i$ , and  $r_i + p_i + p_j \leq V$ , then  $\pi'$  is dominated.

**Rule 22.** If  $ag_i = A, ag_j = B, t \leq r_i$ , and  $r_i + p_i \leq U - p_j$ , then  $\pi'$  is dominated.

**Rule 23.** If  $ag_i = A, ag_j = B, t \leq r_i = U - p_j, i < j$ , and  $r_i + p_i + p_j \leq V$ , let  $\pi'$  be dominated.

**Rule 24.** If  $ag_i = A, ag_j = B, r_i \leq t, U - p_j \leq t, i < j$ , and  $t + p_i + p_j \leq V$ , let  $\pi'$  be dominated.

*Situation 5:* Consider job  $i$  belonging to agent B and job  $j$  belonging to agent A. In Rules 25–28,  $\alpha$  has an earlier completion time, so  $\pi'$  is dominated. In Rules 29–30, although  $\alpha$  is tied with  $\alpha'$  in terms of completion time, we let  $\pi$  dominate  $\pi'$  if  $i < j$ . This situation involves comparing the completion times of jobs belonging to agents B and A.

**Rule 25.** If  $ag_i = B, ag_j = A, U - p_i \leq t < r_j \leq t + p_i$ , and  $t + p_i + p_j \leq d_j$ , then  $\pi'$  is dominated.

**Rule 26.** If  $ag_i = B, ag_j = A, U - p_i \leq t$ , and  $t + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 27.** If  $ag_i = B, ag_j = A, t \leq U - p_i < r_j \leq U$ , and  $U + p_j \leq d_j$ , then  $\pi'$  is dominated.

**Rule 28.** If  $ag_i = B, ag_j = A, t \leq U - p_i$ , and  $U \leq r_j$ , then  $\pi'$  is dominated.

**Rule 29.** If  $ag_i = B, ag_j = A, t \leq U - p_i = r_j, i < j$ , and  $U + p_j \leq d_j$ , let  $\pi'$  be dominated.

**Rule 30.** If  $ag_i = B, ag_j = A, U - p_i \leq t, r_j \leq t, i < j$ , and  $t + p_i + p_j \leq d_j$ , let  $\pi'$  be dominated.

*Situation 6:* Consider job  $i$  belonging to agent A and job  $j$  belonging to agent C. In Rules 31–32,  $\alpha$  has a tied waiting time but an earlier completion time, so  $\pi'$  is still dominated by  $\pi$ . Similarly, in this situation, we observe the mutual influence of jobs belonging to agents A and C.

**Rule 31.** If  $ag_i = A, ag_j = C, r_i \leq t$ , and  $t + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 32.** If  $ag_i = A, ag_j = C, t \leq r_i$ , and  $r_i + p_i \leq r_j$ , then  $\pi'$  is dominated.

*Situation 7:* Consider job  $i$  belonging to agent C and job  $j$  belonging to agent A. In Rules 33–36,  $\alpha$  has not only an earlier completion time but also a shorter waiting time, so  $\pi'$  is dominated. In Rules 37–38,  $\alpha$  has a tied completion time but a shorter waiting time, so  $\pi'$  is still dominated by  $\pi$ . Similar to the previous situation, this time the two jobs belong to agents C and A respectively.

**Rule 33.** If  $ag_i = C, ag_j = A, r_i \leq t < r_j \leq t + p_i$ , and  $t + p_i + p_j \leq d_j$ , then  $\pi'$  is dominated.

**Rule 34.** If  $ag_i = C, ag_j = A, r_i \leq t$ , and  $t + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 35.** If  $ag_i = C, ag_j = A, t \leq r_i \leq r_j \leq r_i + p_i$ , and  $r_i + p_i + p_j \leq d_j$ , then  $\pi'$  is dominated.

**Rule 36.** If  $ag_i = C, ag_j = A, t \leq r_i$ , and  $r_i + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 37.** If  $ag_i = C, ag_j = A, t \leq r_i = r_j$ , and  $r_i + p_i + p_j \leq d_j$ , then  $\pi'$  is dominated.

**Rule 38.** If  $ag_i = C, ag_j = A, r_i \leq t, r_j \leq t$ , and  $t + p_i + p_j \leq d_j$ , then  $\pi'$  is dominated.

*Situation 8:* Consider job  $i$  belonging to agent B and job  $j$  belonging to agent C. In Rules 39–40, although  $\alpha$  has a

tied waiting time, it has an earlier completion time. Hence,  $\pi'$  is still dominated by  $\pi$ . Similarly, this time we observe the interaction of jobs, first belonging to agent B and then the subsequent job coming from agent C.

**Rule 39.** If  $ag_i = B, ag_j = C, U - p_i \leq t$ , and  $t + p_i \leq r_j$ , then  $\pi'$  is dominated.

**Rule 40.** If  $ag_i = B, ag_j = C, t \leq U - p_i$ , and  $U \leq r_j$ , then  $\pi'$  is dominated.

*Situation 9:* Consider job  $i$  belonging to agent C and job  $j$  belonging to agent B. In Rules 41–44,  $\alpha$  has not only an earlier completion time but also a shorter waiting time, so  $\pi'$  is dominated. In Rules 45–46,  $\alpha$  has a tied completion time but a shorter waiting time, so  $\pi'$  is still dominated by  $\pi$ . Finally, we examine the interaction of jobs, first belonging to agent C and then the subsequent job belonging to agent B.

**Rule 41.** If  $ag_i = C, ag_j = B, r_i \leq t < U - p_j \leq t + p_i$ , and  $t + p_i + p_j \leq V$ , then  $\pi'$  is dominated.

**Rule 42.** If  $ag_i = C, ag_j = B, r_i \leq t$ , and  $t + p_i \leq U - p_j$ , then  $\pi'$  is dominated.

**Rule 43.** If  $ag_i = C, ag_j = B, t \leq r_i < U - p_j \leq r_i + p_i$ , and  $r_i + p_i + p_j \leq V$ , then  $\pi'$  is dominated.

**Rule 44.** If  $ag_i = C, ag_j = B, t \leq r_i$ , and  $r_i + p_i \leq U - p_j$ , then  $\pi'$  is dominated.

**Rule 45.** If  $ag_i = C, ag_j = B, t \leq r_i = U - p_j$ , and  $r_i + p_i + p_j \leq V$ , then  $\pi'$  is dominated.

**Rule 46.** If  $ag_i = C, ag_j = B, r_i \leq t, U - p_j \leq t$ , and  $t + p_i + p_j \leq V$ , then  $\pi'$  is dominated.

These dominance rules will be embedded in later branch-and-bound algorithms, serving to exclude unnecessary candidates in a search tree.

## B. LOWER BOUND

In this subsection, we propose a lower bound (named LB) in a preemptive way. Let  $\pi = (\alpha, \beta)$  be a semi-determined schedule; i.e., the former  $l$  jobs form a determined partial sequence  $\alpha$  and the  $(n - l)$  remaining jobs form a set of  $\beta$ .

FIGURE 2 shows the pseudo code of the lower bound. In Step 1, let cook  $c$  process the last job of  $\alpha$ . In Step 2, we prepare each cook's time slots to indicate if he is available or not. In Step 3, we sort the jobs in  $\beta$  in ascending order of release time, agent, and processing time for  $j = l + 1, l + 2, \dots, n$ . Note that we only sort these jobs and do not disrupt their structures or create agreeable jobs artificially. Therefore, the proposed lower bound can more accurately approach the true minimal costs. The job at position  $j$  in the sorted sequence is denoted by job  $(j)$  for  $j = l + 1, l + 2, \dots, n$ . In the following steps, we allocate the jobs belonging to agents A and B first, since they cannot tolerate tardiness. For each job  $(j)$ , we partition it into units and allocate these units to several available cooks as late as possible, i.e., in a preemptive way, if they are available during the period of  $[r_{(j)}, d_{(j)}]$  in Steps 7–8; otherwise, we directly allocate the remaining units to some buffered slots in  $[0, r_{(j)}]$  in Steps 9–10. It is equivalent to shifting previously allocated slots to earlier buffered slots and making new room in  $[r_{(j)}, d_{(j)}]$  for the remaining units of job  $(j)$ . If there are still some unused slots for job  $(j)$  (i.e.,

$d_{(j)} - r_{(j)}$  is much larger than  $p_{(j)}$  in  $[r_{(j)}, d_{(j)}]$ , Step 11 lets them be a buffer for future jobs. So far, we have allocated all the remaining jobs of agents A and B in a preemptive way. Now we can start to estimate the objective cost for agent C and set the initial cost in Step 12. In Steps 16–19, if there is no sufficient room in  $[r_{(j)}, r_{(j)} + p_{(j)}]$  for accommodating job  $(j)$ , we shift some slots containing jobs of agent A and B to earlier buffered slots and make new room for job  $(j)$  belonging to agent C in Steps 17–19. Then we accumulate the cost of each job of agent C in Step 20. In Step 21, let the lower bound be at least the sum of the processing times of agent C's remaining jobs. Lastly, we output the final result in Step 22.

The novelty of this lower bound method lies in two key aspects. First, unlike the approach outlined in [23], we avoid the creation of new virtual jobs to align with existing ones, thus preserving the original structure of each problem instance. Second, we minimize job partitioning unless absolutely necessary. A combined buffer is utilized for the remaining jobs, with job  $i$  initially placed at the rear of this buffer when allocated to agents A or B. Subsequently, if job  $j$  cannot be accommodated due to space constraints, job  $i$  is shifted to the front empty space of its buffer, if available. If space is still insufficient, we partition job  $i$  and distribute its excess to other jobs' buffers. This strategy minimizes job preemption and ensures the problem's structural integrity is maintained, leading to a more accurate lower bound estimation without significant underestimation.

### C. BRANCH-AND-BOUND ALGORITHM

In this section, we propose a branch-and-bound algorithm (named BB) based on these dominance rules and the lower bound. Note that the algorithm is recursively performed in a depth-first-search (DFS) manner

The main program is shown in FIGURE 3. Before calling the recursive algorithm, we initially let  $\pi^*$  be any feasible schedule and  $f^* = f(\pi^*)$ , where  $f()$  means the objective function. Then  $BB(\pi, 0)$  is called for obtaining the optimal schedules. If the current node is at level  $n$ , i.e., a leaf node, we can calculate its objective cost. If a lower cost is found, the optimal cost and the optimal schedule are replaced by the current one (Steps 1–2). If BB is halfway, i.e.,  $l < n$ , and the current search is dominated by any rule or the lower bound, BB will be bounded and not explore this branch further (Steps 3–4). Otherwise, we will prepare  $n - l + 1$  new dummy schedules and explore them recursively. That is, we keep traversing the  $n - l + 1$  child branches in DFS order (Steps 6–9). In the end, all the nodes are either visited or pruned, and the optimal solutions are stored in the two global variables  $f^*$  and  $\pi^*$ .

### V. COMPUTATIONAL EXPERIMENTS

In this section, we aim to compare the performances of different lower bounds. Therefore, for convenience, let  $BB^1$  denote a branch-and-bound algorithm with the simplest lower bound estimated only by summing up the processing times of all the remaining jobs of agent C;  $BB^2$ , a branch-and-bound

algorithm with the lower bound proposed by Shiau, et al. [28]; and  $BB^3$ , a branch-and-bound algorithm with our proposed lower bound shown in FIGURE 2. Moreover, we integrate the three lower bounds into a hybrid one and let  $BB^*$  denote a branch-and-bound algorithm with the hybrid bound. Note that the solution quality of each branch-and-bound algorithm is the same (i.e., the optimal); we compare their performances in terms of nodes and run time. Moreover, we also observe how the number of cooks ( $m$ ), number of jobs ( $n$ ), due window ( $U, V$ ), and release time ( $r_j$ ) influence the performance of  $BB^*$  in greater detail.

TABLE 2 shows all the parameters used in the following experiments. Parameters  $m, n, p_j, d_j, r_j, U$ , and  $V$  are defined as earlier. Each  $p_j$  follows a discrete uniform distribution  $DU(1, 100)$ ;  $U$  follows a discrete uniform distribution  $DU(100, 50n/m)$ . Control parameter  $\gamma$  forces release time  $r_j$  to follow  $DU(0, 100n\gamma/m)$ ; control parameter  $\omega$  forces  $V$  to follow  $DU(U + 100, U + 50 + 10n\omega / m)$ ; control parameters  $\tau$  and  $R$  let due date  $d_j$  follow  $DU(100n(1 - \tau - 0.5R), 100n(1 - \tau + 0.5R))$ . All the related algorithms are implemented in object-oriented Pascal and executed on an Intel Core i7-9700 CPU @ 3.00GHz with 32 GB RAM in a Windows 10 environment. For each later setting, 50 random trials are conducted and recorded. All the execution times are measured in seconds.

A pilot study is conducted initially to contrast the performance of our four branch-and-bound algorithms against other mathematical software. As demonstrated in TABLE 3, we utilize MATLAB to execute a brute force search algorithm and employ its in-built ga solver with standard configurations. We adjust the parameters  $\tau$  and  $R$  for a specific setup characterized by  $m = 2, n = 8, n_1 = 2, n_2 = 2, n_3 = 4, U = 200$ , and  $V = 330$ , to scrutinize their performances. Significantly, our proposed algorithms consistently deliver the optimal results, approximately 0.001 seconds, starkly contrasting with the basic MATLAB algorithm which demands at least 10 seconds. On another note, for this generic solver dependent solely on basic crossover and mutation operations, struggles to approximate the optimal solutions, consequently incurring a time cost that exceeds that of the brute force algorithm. The quality of solutions provided by this solver is quantified by the relative error percentage (REP), calculated as  $100\% \times (\text{Cost}_{ga} - \text{Cost}_{BB^*}) / \text{Cost}_{BB^*}$ . It is evident that the solver is inept at solving our problem in its default nonlinear integer programming manner. In sum, these observations accentuate the needs of tailored algorithms for optimally solving our problem with a larger problem size, e.g.,  $n = 25$ .

TABLE 4 shows the performances of the four branch-and-bound algorithms for  $n = 10$ . Note that a branch-and-bound algorithm always generates the optimal solutions. Therefore,  $BB^*$  outperforms the other three in terms of execution speed. That is, all of them have the same solution quality, where column NU means the number of unsolvable instances within 100 million nodes. Since  $BB^1$  is equipped with the simplest lower bound, i.e., limited ability, it must take much run time to traverse a search tree. On the other hand,  $BB^2$  adopts

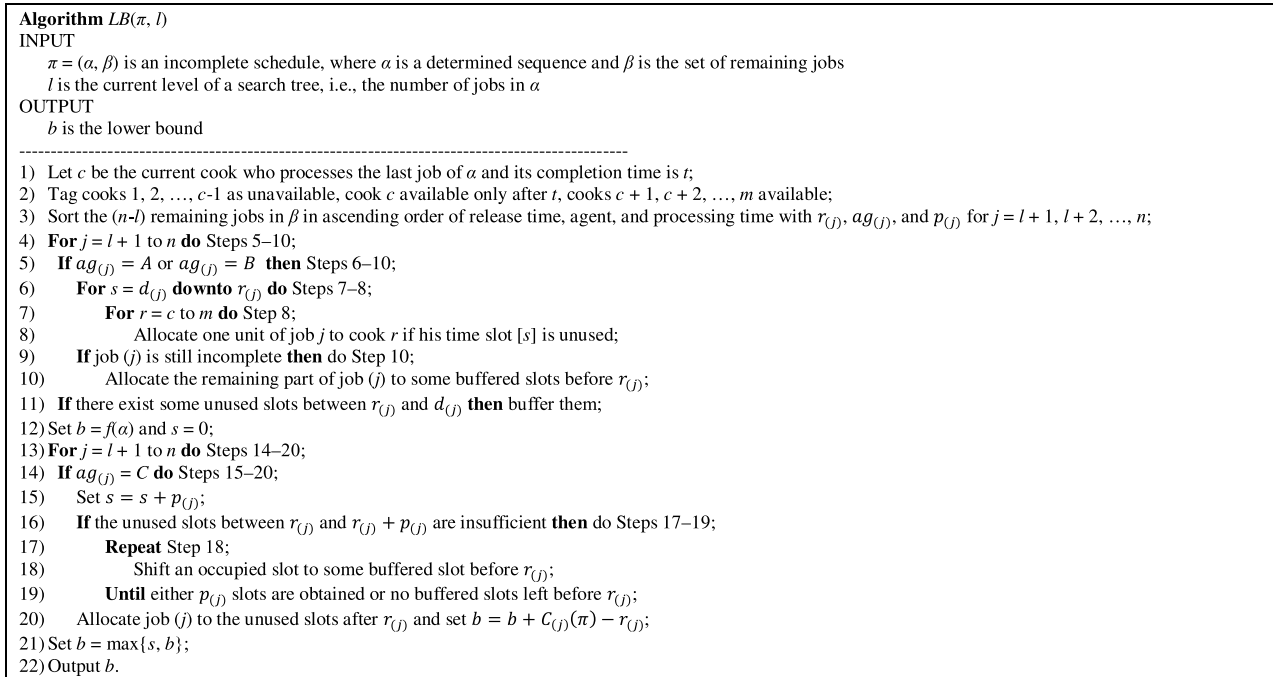


FIGURE 2. The proposed lower bound.

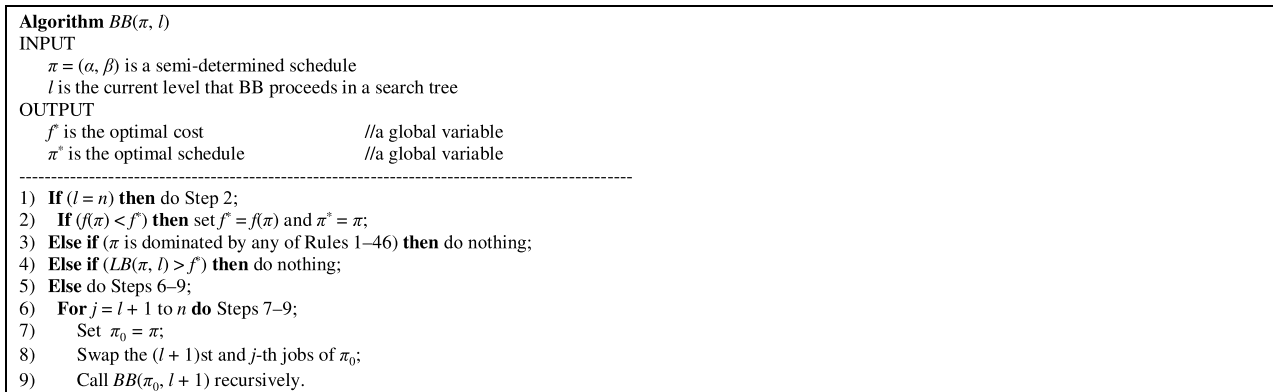


FIGURE 3. The proposed lower bound.

TABLE 2. The related parameters used in this section.

Parameter	Default	Range	Meaning
$m$	3	2, 3, 5	the number of cooks
$n$	20	10, 15, 20, 25	the number of jobs
$n_i$			the number of jobs belonging to cook $i$
$p_j$		DU(1, 100)	the processing time of job $j$
$\gamma$	0.5	0.25, 0.5, 0.75	a control parameter for release time
$r_j$		DU(0, 100n $\gamma$ /m)	the release time of job $j$
$\tau$	0.5	0.25, 0.5, 0.75	a control parameter for due date
$R$	0.5	0.25, 0.5, 0.75	a control parameter for due date
$d_j$		DU( $r_j + 100n(1-\tau-0.5R)$ , $r_j + 100n(1-\tau + 0.5R)$ )	the due date of job $j$
$\omega$	0.5	0.25, 0.5, 0.75	a control parameter for due window $[U, V]$
$U$		DU(100, 50n/m)	the beginning of due window $[U, V]$
$V$		DU( $U + 100, U + 50 + 10n\omega$ /m)	the end of due window $[U, V]$

a re-structured strategy; i.e., the processing times, release times, and due dates are all re-sorted before estimating lower bounds. Such re-organization usually leads to underestimated lower bounds. Moreover, such a re-arrangement is more suitable for single-machine (or *single-cook* in the context of the

presented problem) scheduling. For multi-machine scheduling, after re-sorting processing times, release times, and due dates, each machine will take a small and early job if it can. This will also cause underestimation. By contrast,  $BB^3$  will preempt a job only as a last resort. Therefore, the lower

TABLE 3. The comparison between our algorithms and other software for  $m = 2$  and  $n = 8$ .

		Branch-and-bound algorithms				MATLAB 2023a		
		BB*	BB <sup>1</sup>	BB <sup>2</sup>	BB <sup>3</sup>	Brute force	ga solver	
$\tau$	$R$	Run time	Run time	Run time	Run time	Run time	Run time	REP
0.25	0.25	0.001	0.001	0.001	<0.001	10.694	23.796	15.97%
	0.5	0.001	<0.001	0.001	<0.001	10.692	23.114	11.80%
	0.75	<0.001	<0.001	<0.001	<0.001	10.677	23.721	7.92%
0.5	0.25	<0.001	0.001	0.002	<0.001	10.737	22.299	9.62%
	0.5	<0.001	0.001	0.002	<0.001	10.755	22.852	7.39%
	0.75	<0.001	0.002	0.002	<0.001	10.738	23.642	20.81%
0.75	0.25	0.001	0.001	0.001	<0.001	10.787	26.109	20.24%
	0.5	<0.001	0.002	0.001	0.001	10.844	22.795	20.69%

TABLE 4. The performances of four algorithms for  $m = 3$  and  $n = 10$ .

		BB*			BB <sup>1</sup>			BB <sup>2</sup>			BB <sup>3</sup>		
$\tau$	$R$	Node	Run time	NU	Node	Run time	NU	Node	Run time	NU	Node	Run time	NU
0.25	0.25	4954.04	0.01	0	275252.38	0.14	0	183209.06	0.13	0	6254.78	0.02	0
	0.50	4514.20	0.01	0	252031.36	0.13	0	175649.20	0.13	0	6069.36	0.02	0
	0.75	8694.90	0.02	0	279813.52	0.14	0	191866.98	0.14	0	9917.72	0.02	0
0.50	0.25	10047.86	0.02	0	260661.04	0.13	0	181756.94	0.13	0	11905.58	0.02	0
	0.50	8187.94	0.02	0	265613.60	0.13	0	198115.98	0.14	0	9630.82	0.02	0
	0.75	7813.34	0.02	0	216634.88	0.11	0	152721.62	0.11	0	9077.86	0.02	0
0.75	0.25	3833.68	0.01	0	139260.90	0.07	0	92265.98	0.07	0	5262.10	0.01	0
	0.50	2088.74	0.01	0	72678.86	0.04	0	41322.04	0.04	0	3053.10	0.01	0

TABLE 5. The performances of four algorithms for  $m = 3$  and  $n = 15$ .

		BB*			BB <sup>1</sup>			BB <sup>2</sup>			BB <sup>3</sup>		
$\tau$	$R$	Node	Run time	NU	Node	Run time	NU	Node	Run time	NU	Node	Run time	NU
0.25	0.25	208.80	0.02	0	29196849.67	19.17	38	31537018.35	37.24	33	217.68	0.02	0
	0.50	1548466.42	5.30	0	43642978.56	28.57	41	31012323.18	34.96	39	2005147.34	5.78	0
	0.75	75.80	0.01	0	38755722.60	24.03	45	36073817.13	39.20	42	83.04	0.02	0
0.50	0.25	947360.76	2.54	0	29361931.50	17.91	44	31756811.22	35.60	41	1139963.52	2.58	0
	0.50	283436.82	0.77	0	20374954.13	13.31	42	31186319.15	35.91	37	369794.26	0.83	0
	0.75	2186051.02	4.89	0	50784566.60	30.24	40	37866785.67	34.23	38	2616744.18	5.14	0
0.75	0.25	557917.68	1.50	0	19556234.81	12.92	34	22993975.39	26.95	27	867115.82	1.92	0
	0.50	105801.80	0.29	0	31567256.62	22.26	24	23380224.83	31.87	14	150848.34	0.35	0

TABLE 6. The performances of four algorithms for  $m = 3$  and  $n = 20$ .

		BB*			BB <sup>1</sup>			BB <sup>2</sup>			BB <sup>3</sup>		
$\tau$	$R$	Node	Run time	NU	Node	Run time	NU	Node	Run time	NU	Node	Run time	NU
0.25	0.25	112.50	0.05	0	-	-	50	-	-	50	122.84	0.06	0
	0.50	113.92	0.05	0	41250400.00	88.84	49	56661182.00	179.69	49	125.42	0.06	0
	0.75	113.00	0.05	0	-	-	50	-	-	50	124.56	0.06	0
0.50	0.25	112.88	0.04	0	-	-	50	-	-	50	122.98	0.04	0
	0.50	113.54	0.04	0	-	-	50	-	-	50	125.52	0.04	0
	0.75	113.98	0.04	0	-	-	50	96372576.00	212.67	49	126.64	0.04	0
0.75	0.25	113.50	0.02	0	-	-	50	-	-	50	124.18	0.03	0
	0.50	106.02	0.02	0	7616649.00	15.14	49	9527335.00	32.17	49	115.62	0.02	0

bounds obtained by BB<sup>3</sup> will not be underestimated much. For traditional multi-machine tardiness minimization problems, their problem sizes for a branch-and-bound algorithm are usually not larger than 20. That is, the proposed lower bound achieves tight lower bounds while retaining execution efficiency.

TABLES 5 and 6 show the limits of abilities of BB<sup>1</sup> and BB<sup>2</sup>. Intrinsicly, the presented problem is an NP-hard total completion time minimization problem for multiple cooks; hence,  $n = 20$  is a reasonable problem size that an exact algorithm, e.g., a branch-and-bound algorithm [28], can solve within a given time, e.g., 3600 seconds. Consequently, for the presented problem, most instances of 20 jobs cannot be optimally solved BB<sup>1</sup> within 100 million nodes. Furthermore,

BB<sup>2</sup> cannot solve all the 20-job instances either. The reason is that the lower bound proposed by [28] needs to re-sort and re-assign the remaining due dates, release times, and processing times for each job; i.e., it must use a destructive way. Evidently, this re-structured strategy for designing a lower bound cannot trim unnecessary nodes in the presented problem as early as possible. On the other hand, a closer look at BB<sup>3</sup> reveals that, although the proposed lower bound can prune a lot of unnecessary nodes, it cannot prune all the unnecessary nodes. It implies that such a powerful lower bound still needs additional aid. That is, BB\* is superior to other branch-and-bound algorithms.

FIGURES 4 and 5 illustrate the performance of our algorithm, BB<sup>3</sup>. From the data previously presented in



TABLE 7. The performance of BB\* for  $m = 3$  and  $n = 25$ .

$\tau$	$R$	BB*		
		Node	Run time	NU
0.25	0.25	148.66	0.15	0
	0.50	151.76	0.15	0
	0.75	145.90	0.13	0
0.50	0.25	150.36	0.11	0
	0.50	151.84	0.11	0
	0.75	149.60	0.10	0
0.75	0.25	142.76	0.06	0
	0.50	519948.39	9.51	4

TABLE 8. The performance of BB\* for  $n = 20$ .

$m$	$U$	$V$	$\tau$	$R$	BB*		
					Node	Run time	NU
2	500	750	0.25	0.25	99.72	0.04	0
				0.50	99.86	0.04	0
				0.75	98.30	0.04	0
			0.50	0.25	99.44	0.03	0
				0.50	100.86	0.03	0
				0.75	101.40	0.03	0
			0.75	0.25	397.57	0.03	1
				0.50	1221841.04	21.50	0
				0.75	110.48	0.05	0
			3	333	516	0.25	0.25
0.50	110.18	0.05					0
0.75	114.40	0.05					0
0.50	0.25	113.24				0.04	0
	0.50	112.34				0.04	0
	0.75	112.22				0.04	0
0.75	0.25	110.04				0.02	0
	0.50	111.36				0.02	0
	0.75	138.64				0.07	0
5	200	330				0.25	0.25
			0.50	145.50	0.07		0
			0.75	144.24	0.05		0
			0.50	0.25	140.46	0.05	0
				0.50	146.96	0.05	0
				0.75	147.82	0.03	0
			0.75	0.25	140.94	0.03	0
				0.50	140.94	0.03	0

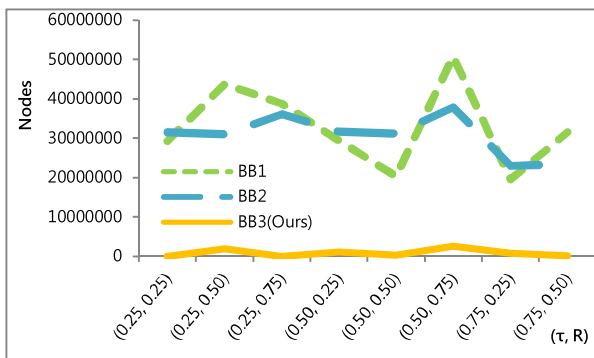


FIGURE 4. The curves of the average nodes in TABLE 5.

TABLE 5, it is evident that the performance of BB<sup>3</sup> is closely comparable to that of BB\*, with almost overlapping results. This indicates that our lower bound (BB<sup>3</sup>) or the hybrid branch-and-bound algorithm BB\*, which incorporates our lower bound, demonstrates significantly lower nodes and run times compared to other lower bounds. Therefore, BB<sup>3</sup> and BB\* can serve as benchmarks for evaluating other meta-heuristic algorithms in future studies.

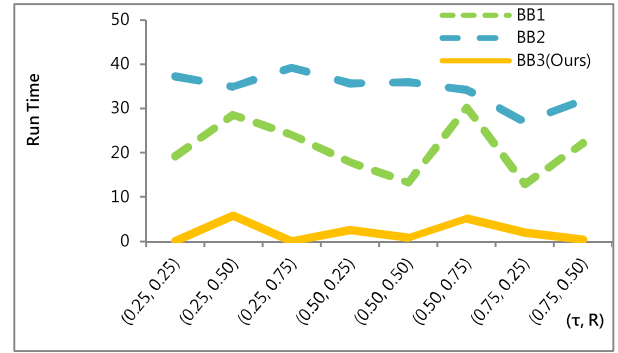


FIGURE 5. The curves of the average run times in TABLE 5.

TABLE 9. The performance of BB\* for  $m = 3$  and  $n = 20$ .

$n_1$	$n_2$	$n_3$	BB*		
			Node	Run time	NU
0	0	20	120.26	0.01	0
0	6	14	133.52	0.01	0
0	12	8	161.73	0.02	2
0	18	2	4875177.50	39.43	42
6	0	14	101.36	0.02	0
6	6	8	105.94	0.02	0
6	12	2	1588001.63	24.00	20
12	0	8	86.48	0.02	0
12	6	2	95.79	0.04	16
18	0	2	83.77	0.04	11

TABLE 7 shows the ability of BB\* for  $n = 25$ . BB\* can solve most problem instances within 10 minutes, i.e., 100 million nodes. However, for a situation of jobs with early due dates (i.e.,  $\tau \geq 0.75$ ), all three agents wish to complete their jobs in an early time period. The temporary objective cost of each partial schedule is similar to those of others; that is, semi-complete schedules are similar and cannot be distinguished as early as possible, and more explorations of a search tree are needed. Moreover, early and scattered due dates (i.e., large  $R$ ) also make the situations worse. It therefore needs to visit more nodes and cannot generate the optimal solutions within 100 million nodes. Consequently,  $n = 20$  is a proper problem size that BB\* can optimally solve within 10 minutes for a real-world problem instance.

TABLE 8 shows the effect of  $m$  on the performance of BB\*. Note that there are 20 jobs. A small number of cooks (i.e.,  $m \leq 3$ ) means intense resource competition. Even if cook 1 is assigned 8 jobs, we still cannot determine if such a partial schedule is worth keeping or not. This uncertainty implies that more trial and error runs are still needed to prune or retain it. Consider an extreme example and suppose there are 20 cooks. Then each cook is intuitively assigned only one job, and we can obtain an optimal schedule easily, i.e., less run time. That is why a 5-cook instance will take less run time. Or consider that cook 1 is assigned 8 jobs in a 20-job-and-5-cook instance; it will almost certainly determine that it is a bad partial schedule. The reason is that each cook is most likely to be assigned 4 or 5 jobs in an optimal schedule. In this example, cook 1 is overloaded, so we can prune this bad partial schedule as early as possible when  $m = 5$ .

TABLE 10. The performance of BB\* for  $m = 3$  and  $n = 20$ .

$\gamma$	$\tau$	$R$	BB*		
			Node	Run time	NU
0.25	0.25	0.25	93.14	0.03	0
		0.50	95.50	0.03	0
		0.75	93.36	0.03	0
0.50	0.25	0.25	95.90	0.03	0
		0.50	95.76	0.03	0
		0.75	93.64	0.03	0
0.75	0.25	0.25	92.84	0.01	0
		0.50	90.30	0.01	0
		0.75	90.30	0.01	0
0.50	0.25	0.25	110.90	0.05	0
		0.50	110.22	0.05	0
		0.75	113.42	0.05	0
0.50	0.25	0.25	112.94	0.04	0
		0.50	112.10	0.04	0
		0.75	110.76	0.04	0
0.75	0.25	0.25	108.44	0.02	0
		0.50	108.84	0.02	1
		0.75	108.84	0.02	1
0.75	0.25	0.25	89.30	0.03	0
		0.50	92.74	0.04	0
		0.75	94.88	0.04	0
0.50	0.25	0.25	93.12	0.03	0
		0.50	89.60	0.02	0
		0.75	92.44	0.03	0
0.75	0.25	0.25	93.66	0.02	0
		0.50	93.28	0.02	0

In TABLE 9, we observe the performance of BB\* if the jobs of three agents vary with amount. In general, BB\* will spend much execution time if three agents have equal amounts of jobs. If agent B does not have any jobs, the problem degenerates into a two-agent scheduling problem, i.e., a simpler problem. On the other hand, if  $n_2$  is more than half of  $n$ , it is difficult to cram agent B's jobs into a fixed length due window  $[U, V]$ . Instead, a large  $n_3$  will not make the problem more difficult. This implies that constraints such as due window and zero tardiness mainly dominate the execution time.

TABLE 10 shows the effect of release time on the performance of BB\*. A small  $\gamma$  means early release times. All the jobs are released at the beginning of a whole makespan, and thus BB\* has greater flexibility to schedule them. On the other hand, a large  $\gamma$  means that all the jobs are uniformly released in a whole makespan. For those jobs released in the rear time period, we have little chance to adjust their schedules, even if all the cooks are idle during the front time period. Moreover, early and scattered due dates (i.e., large  $\tau$  and  $R$ ) also make the situation worse. For a decision maker, he should determine release times as early as possible and request later due dates as much as he can. If so, BB\* can generate the optimal solutions in time.

TABLE 11 shows the influence of due window on the performance of BB\*. A small  $\omega$  means a narrow due window. In general, a narrow due window increases the amount of computation needed for BB\* to solve the problem. Compressing all the completion times of agent B's jobs into a small due window  $[U, V]$  yields many similar partial schedules. It is hard to tell if a partial schedule is worth exploring further. Consequently, a narrow due window may exhaust all the

TABLE 11. The performance of BB\* for  $m = 3$  and  $n = 20$ .

$\omega$	$U$	$V$	$\tau$	$R$	BB*			
					Node	Run time	NU	
0.25	333	449	0.25	0.25	112.31	0.05	2	
					0.50	114.65	0.05	2
					0.75	117.20	0.05	0
0.50	333	449	0.25	0.25	115.88	0.03	1	
					0.50	116.22	0.04	0
					0.75	115.52	0.04	2
0.75	333	449	0.25	0.25	111.96	0.02	3	
					0.50	1385092.28	9.22	4
					0.75	110.04	0.05	0
0.50	333	516	0.25	0.25	109.76	0.05	0	
					0.50	109.76	0.05	0
					0.75	114.14	0.05	0
0.50	333	516	0.25	0.25	112.34	0.04	0	
					0.50	112.00	0.04	0
					0.75	111.16	0.04	0
0.75	333	516	0.25	0.25	110.26	0.02	0	
					0.50	860164.26	8.10	0
					0.75	107.20	0.05	0
0.75	333	583	0.25	0.25	109.68	0.05	0	
					0.50	109.68	0.05	0
					0.75	113.94	0.06	0
0.50	333	583	0.25	0.25	109.70	0.04	0	
					0.50	107.44	0.04	0
					0.75	113.52	0.04	0
0.75	333	583	0.25	0.25	117.76	0.03	0	
					0.50	111.10	0.02	0

manpower for agent B and deter all the cooks from processing other agents' jobs during this period.

In this section, BB\* outperforms the other branch-and-bound algorithms in terms of execution speed. This superior performance can be attributed to the proposed hybrid lower bound. Since it retains most structures of the jobs, its resultant lower bound is able to approach the corresponding minimal objective cost and will not exceed it.

## VI. CONCLUSION

In this study, we present an interesting multi-agent scheduling problem, highlighting the significant imbalance in resource allocation between the real and virtual worlds. This imbalance is particularly noticeable in environments where customer interactions span both digital and physical spaces, such as restaurants on platforms like Uber Eats or retail stores with both online and dine-in services. Our model addresses these challenges using three-agent scheduling strategies that optimize the distribution of resources, ensuring that no customer segment is disproportionately disadvantaged.

Our model employs three-agent scheduling strategies by effectively preempting some urgent jobs from agents A and B, while also striving to minimize the total waiting time for agent C. This model not only balances the immediate demands of virtual interactions but also respects the needs of real-world customers, who often suffer from longer waiting times due to the prioritization of online ratings. By preempting some large jobs, we significantly improve both service efficiency and customer satisfaction across all agents.

While this exact algorithm can handle different constraints and objectives from three distinct agents simultaneously, it faces limitations, particularly when dealing with larger

problem sizes. The proposed branch-and-bound algorithm, while effective, becomes very time-consuming when the problem size grows beyond a certain threshold. This limitation highlights the necessity for further enhancements in some approximate algorithms, particularly to expand the scalability of three-agent scheduling for larger problem instances.

The insights gained from this study could be applied to other industries facing similar challenges with multiple requirements from multiple agents. Industries such as health-care, logistics, and manufacturing, which often grapple with complex and multifaceted scheduling needs, could benefit significantly from the scheduling strategies developed here. Looking forward, we aim to adapt these strategies to metaheuristic algorithms, which are particularly suited to scenarios exceeding problem sizes of 25, where real-time and near-optimal solutions are essential. We plan to develop metaheuristic algorithms, such as genetic algorithm, to deal with larger-scale real-world instances. Furthermore, we intend to use the exact algorithm as a benchmark to assess the effectiveness of these metaheuristic solutions in complex multi-agent environments.

## REFERENCES

- [1] Q. P. Tan, L. Huang, D. Xu, Y. Cen, and Q. Cao, "Serious game for VR road crossing in special needs education," *Electronics*, vol. 11, no. 16, p. 2568, Aug. 2022.
- [2] K.-L. Hsiao, M. D. Lytras, and C.-C. Chen, "An in-app purchase framework for location-based AR games: The case of Pokémon go," *Library Hi Tech*, vol. 38, no. 3, pp. 638–653, Jun. 2019.
- [3] C.-H. Su and J.-Y. Wang, "A makespan minimization problem for versatile developers in the game industry," *RAIRO Oper. Res.*, vol. 56, no. 6, pp. 3895–3913, Nov. 2022.
- [4] K.-C. Yao, J.-J. Yang, H.-W. Lo, S.-W. Lin, and G.-H. Li, "Using a BBWM-PROMETHEE model for evaluating mobile commerce service quality: A case study of food delivery platform," *Res. Transp. Bus. Manage.*, vol. 49, Aug. 2023, Art. no. 100988.
- [5] J.-Y. Wang, "Minimizing the average waiting time of unequal-size data items in a mobile computing environment," *Mobile Inf. Syst.*, vol. 2016, pp. 1–14, Jan. 2016.
- [6] E. Pastore and A. Alfieri, "An effective approach for total completion time minimization subject to makespan constraint in permutation flowshops," *Eng. Optim.*, pp. 1–6, Feb. 2024.
- [7] V. R. Vijayalakshmi, M. Schröder, and T. Tamir, "Minimizing total completion time with machine-dependent priority lists," *Eur. J. Oper. Res.*, vol. 315, no. 3, pp. 844–854, Jun. 2024.
- [8] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici, "Scheduling problems with two competing agents," *Oper. Res.*, vol. 52, no. 2, pp. 229–242, Apr. 2004.
- [9] K. R. Baker and J. C. Smith, "A multiple-criterion model for machine scheduling," *J. Scheduling*, vol. 6, no. 1, pp. 7–16, 2003.
- [10] W.-C. Lee, Y.-H. Chung, and J.-Y. Wang, "A parallel-machine scheduling problem with two competing agents," *Eng. Optim.*, vol. 49, no. 6, pp. 962–975, Jun. 2017.
- [11] T. Ren, Z. Dong, F. Qi, J. Weng, and H. Xue, "Solving the flow-shop scheduling problem with human factors and two competing agents with deep reinforcement learning," *Eng. Optim.*, pp. 1–17, Apr. 2024.
- [12] J. Phosavanh and D. Oron, "Two-agent single-machine scheduling with a rate-modifying activity," *Eur. J. Oper. Res.*, vol. 312, no. 3, pp. 866–876, Feb. 2024.
- [13] S.-S. Li and R.-X. Chen, "Competitive two-agent scheduling with release dates and preemption on a single machine," *J. Scheduling*, vol. 26, no. 3, pp. 227–249, Jun. 2023.
- [14] M. Avolio, "Balancing the average weighted completion times in large-scale two-agent scheduling problems: An evolutionary-type computational study," *Mathematics*, vol. 11, no. 19, p. 4034, Sep. 2023.
- [15] V. T'kindt, F. Della Croce, and A. Agnetis, "Single machine adversarial bilevel scheduling problems," *Eur. J. Oper. Res.*, vol. 315, no. 1, pp. 63–72, May 2024.
- [16] D. Li, G. Li, and F. Cheng, "Two-agent single machine scheduling with deteriorating jobs and rejection," *Math. Problems Eng.*, vol. 2022, pp. 1–10, Nov. 2022.
- [17] C. He, H. Lin, and X. Han, "Two-agent scheduling on a bounded series-batch machine to minimize makespan and maximum cost," *Discrete Appl. Math.*, vol. 322, pp. 94–101, Dec. 2022.
- [18] R. Chen, Z. Geng, L. Lu, J. Yuan, and Y. Zhang, "Pareto-scheduling of two competing agents with their own equal processing times," *Eur. J. Oper. Res.*, vol. 301, no. 2, pp. 414–431, Sep. 2022.
- [19] M. Avolio and A. Fuduli, "A Lagrangian heuristics for balancing the average weighted completion times of two classes of jobs in a single-machine scheduling problem," *EURO J. Comput. Optim.*, vol. 10, Jan. 2022, Art. no. 100032.
- [20] A. Azerine, M. Boudhar, and D. Rebaine, "A two-machine no-wait flow shop problem with two competing agents," *J. Combinat. Optim.*, vol. 43, no. 1, pp. 168–199, Jan. 2022.
- [21] L. Wan, J. J. Mei, and J. Z. Du, "Two-agent scheduling of unit processing time jobs to minimize total weighted completion time and total weighted number of tardy jobs," *Eur. J. Oper. Res.*, vol. 290, no. 1, pp. 26–35, 2021.
- [22] Y. Zhang, J. Yuan, C. T. Ng, and T. C. E. Cheng, "Pareto-optimization of three-agent scheduling to minimize the total weighted completion time, weighted number of tardy jobs, and total weighted late work," *Nav. Res. Logistics (NRL)*, vol. 68, no. 3, pp. 378–393, Apr. 2021.
- [23] M. Haouari, A. Kooli, E. Néron, and J. Carlier, "A preemptive bound for the resource constrained project scheduling problem," *J. Scheduling*, vol. 17, no. 3, pp. 237–248, Jun. 2014.
- [24] R. Lin, J.-Q. Wang, and A. Oulamara, "Online scheduling on parallel-batch machines with periodic availability constraints and job delivery," *Omega*, vol. 116, Apr. 2023, Art. no. 102804.
- [25] C.-H. Su and J.-Y. Wang, "A branch-and-bound algorithm for minimizing the total tardiness of multiple developers," *Mathematics*, vol. 10, no. 7, p. 1200, Apr. 2022.
- [26] J.-Y. Wang, "Minimizing the total weighted tardiness of overlapping jobs on parallel machines with a learning effect," *J. Oper. Res. Soc.*, vol. 71, no. 6, pp. 910–927, Jun. 2020.
- [27] W. Lei, L. Sun, N. Ren, X. Jia, and J.-B. Wang, "Research on delivery times scheduling with sum of logarithm processing times-based learning effect," *Asia-Pacific J. Oper. Res.*, vol. 41, no. 2, Apr. 2024, Art. no. 2350014.
- [28] Y.-R. Shiau, W.-C. Lee, Y.-S. Kung, and J.-Y. Wang, "A lower bound for minimizing the total completion time of a three-agent scheduling problem," *Inf. Sci.*, vols. 340–341, pp. 305–320, May 2016.



**YU-CHUAN CHEN** received the Ph.D. degree in information management from the National Yunlin University of Science and Technology, Taiwan, in 2011. He is currently an Assistant Professor with the Department of Intelligent Technology and Application, Hungkuang University, Taiwan. His research interests include information management systems, artificial intelligence, big data, the Internet of Things, and business intelligence.



**JEN-YA WANG** received the Ph.D. degree in computer science and engineering from National Chung Hsing University, Taiwan, in 2009. He is currently an Associate Professor with the Department of Information Management, National Taichung University of Science and Technology, Taiwan. His research interests include optimization algorithms, database systems, patent search-ing, medical imaging, and artificial intelligence.

• • •