Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 553–563
DOI: 10.15439/2025F6496
ISSN 2300-5963 ACSIS, Vol. 43

The Power of Preemptions in Scheduling on Shareable Resources

Omer Lapidot, Tami Tamir
0009-0002-8688-7687
0000-0002-8409-562X
School of Computer Science
Reichman University, Herzliya, Israel
Email: omer2372@gmail.com, tami@runi.ac.il

***Abstract*—Many combinatorial optimization problems arise in the context of resource allocation. In this paper, we study the problem of allocating shareable resources of different types to jobs, where each job consists of multiple tasks, and each task has a demand of a given duration to a single resource type. All resources are available over a common time interval. Several copies from each resource may be allocated. In a valid solution, at any given time, each job may be processed by at most one resource, and each resource may process at most one job. The objective is to complete all jobs while minimizing the total cost of the allocated resources.**

We focus on the power of preemptions in this model, analyzing how much the total cost can be reduced when jobs are allowed to be preempted — that is, when the processing of tasks can be split into multiple intervals.

We present both theoretical and experimental results, distinguishing between environments where jobs may be preempted but all intervals of a task must be processed on the same resource copy (weak preemptions), and environments where jobs may split the processing of a task among different resource copies (strong preemptions). Without preemptions, the problem is clearly NP-hard, as it generalizes the classical Bin Packing problem. We provide an optimal polynomial-time algorithm for the strong-preemption model, as well as a polynomial-time algorithm for the non-preemption model under a restricted class of task durations. Our empirical evaluation investigates the performance of several greedy heuristics, showing that even simple methods can achieve near-optimal results.

I. Introduction

SUSTAINABILITY plays a crucial role in various aspects of our future on Earth. Growing awareness of the need to protect the environment, combined with the ability to communicate easily, has led to new trends in resource consumption. Users have become more flexible about consumption and are willing and able to share resources in new ways. The new ways to consume resources give rise to new combinatorial problems related to allocation and utilization of shareable resources. In this work, we define and study problem of this kind.

We consider a *resource sharing* model, where each resource is capable of being used or leased for any duration within its availability window. For example, consider Amazon drivers who perform deliveries in their assigned regions, using trucks of different sizes tailored to specific delivery needs. A similar setting arises in large organizations, where departments might share meeting rooms of various capacities, or in workplaces where part-time or hybrid employees share desks, cubicles, or workstations. Likewise, in residential communities, neighbors may collaboratively use equipment such as electric vehicle charging stations [18], lawn mowers or other landscaping tools.

In our resource sharing problem, every user (job) is associated with a demand for several resources. For every user and resource, it is known for how long the user needs the resource. Without sharing, every user is allocated one copy from each resource. However, different users can share the same resource if they use it at different times. Assume that every copy of a resource is associated with a given cost. The goal is to schedule the users on different copies of the resources such that the total cost of allocated resources is minimal. In the basic setting of this model, every user is using each resource consequently and with no interruptions. When preemptions are allowed, a user may split the use of a resource into several segments. It is well known that in many scheduling problems, preemptions significantly affect the objective function value and the computational complexity of the problem. For example, the classical load balancing problem of minimizing the completion time of a schedule (the makespan) is NP-hard if preemptions are not allowed [5] and is solved by a linear-time algorithm if preemptions are allowed

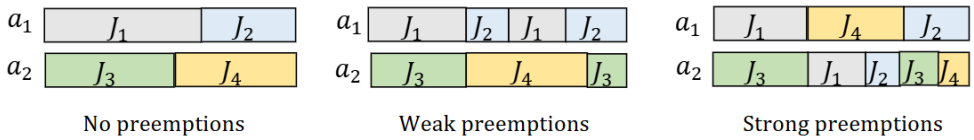

Fig. 1. Possible schedules of four jobs on two copies of resource type a.

[11]. In this work, we study the power of preemption in the resource sharing model. Practically, the power of preemption measures how the flexibility of users in the way they are ready to use the resource affects the total cost.

The resource sharing problem is dual to *open-shop scheduling*, in which we have one copy of each resource type, and the goal is to minimize the schedule's completion time (the makespan). For a single resource type, our problem is identical to the well-known NP-hard *bin-packing* problem in which items of different sizes need to be packed in a minimal number of identical bins. In the bin-packing problem, if items can split among several bins, the problem becomes trivial. In our setting, such splits correspond to job-preemptions. Thus, our problem can be viewed as a generalization of bin-packing. In this generalization with several resource types, the power of preemption has not been studied in the past.

In our analysis we distinguish between four types of resource consumption: (i) no resource sharing, (ii) resource sharing without preemptions, (iii) resource sharing with weak preemptions, where the processing of a task can be paused and resumed on the same resource copy, and (iv) resource sharing with strong preemptions, where tasks may pause and resume, and also migrate across different resource copies. We present an optimal algorithm for the problem with strong preemptions allowed, and analyze theoretically and by experiments the performance of some natural greedy heuristics, in general and for restricted classes of instances. As we show, even simple heuristics can achieve near-optimal performance.

A. Notation and Problem Statement

An instance of the *resource sharing* problem is given by a tuple $\langle \mathcal{J}, M, (p_{ij})_{i\in M, j\in\mathcal{J}}, (c_i)_{i\in M}, T\rangle$, where $\mathcal{J}$ is a set of $n \geq 1$ jobs, M is a set of $m \geq 1$ resource-types, p is an $n \times m$ matrix of processing requests, $c_i > 0$, is the cost of each copy of resource i, and $[0, T]$ is the availability interval of each of the resource copies. We use the term resource i to denote resource-type i. Each job $j \in \mathcal{J}$ consists of several tasks. Formally, each job is associated with a set of resource types $S_j \subseteq M$ corresponding to the tasks composing it (for which $p_{ij} > 0$). For each $i \in S_j$, job j should be processed on a resource of type i for p_{ij} time units. In a valid instance $\sum_i p_{ij} \leq T$. Let $J_i = |\{j \in \mathcal{J} \mid i \in S_j\}|$ be the number of jobs that need resource i.

The idea in the sharing model is that several jobs can share a single copy of a resource, as long as each copy services at most one job in each time slot. Therefore, the number of copies required from resource i may be lower than J_i. It is easy to see that the actual number of copies required from resource i is between $\left\lceil \sum_j p_{ij}/T \right\rceil$ and J_i.

A *non-preemptive schedule* of an instance is denoted by $\sigma = (\sigma_j)_{j\in\mathcal{J}}$. For every job j, $\sigma_j = (\sigma_{ji})_{i\in S_j}$ is the schedule of job j, given by the set of intervals in which j uses the resources it needs. Specifically, $\sigma_{ji} = [start(\sigma_{ji}), end(\sigma_{ji}))$ is the interval in which job j uses resource i in σ. In a feasible solution, $end(\sigma_{ji}) - start(\sigma_{ji}) = p_{ij}$; that is, job j is assigned to the resource for p_{ij} time units. The intervals in which a job uses different resources must be disjoint; that is, for all j and any i_1, i_2, it holds that $\sigma_{ji_1} \cap \sigma_{ji_2} = \emptyset$.

For a given schedule σ, for any time $0 \leq t \leq T$, let $N_i(\sigma, t) = |\{j \in \mathcal{J} \mid t \in \sigma_{ji}\}|$ be the number of jobs $j \in \mathcal{J}$ that use resource i at time t. Let $N_i(\sigma) = \max_{0\leq t<T} N_i(\sigma, t)$. That is, $N_i(\sigma)$ is the maximal number of jobs who simultaneously use resource i throughout the schedule. It holds that $N_i(\sigma)$ is the number of copies of resource i needed to satisfy all job requirements. The cost of a schedule σ is the total cost of the resources used in σ, given by $cost(\sigma) = \sum_{i=1}^{m} N_i(\sigma) \cdot c_i$. The goal is to satisfy all job requirements at a minimum total resource cost, i.e., find a schedule σ such that $cost(\sigma)$ is minimal.

When preemptions are allowed, the processing of a task on one resource may split into several disjoint intervals. We distinguish between two types of preemptions (see Figure 1).

1) Weak Preemptions - for every job j and type $i \in S_j$, the processing of the task of type i of job j may split into several intervals, all of them are on the same copy of a resource of type i.
2) Strong Preemptions - for every job j and type $i \in S_j$, the processing of the task of type i of job j may split into several intervals, which can be processed on different copies of resources of type i.

For an instance I, let $OPT_{nopmtn}(I)$, $OPT_{Wpmtn}(I)$, $OPT_{Spmtn}(I)$ denote the minimal cost of a feasible schedule of I without preemptions, with weak preemptions, and with strong preemptions, respectively.

Definition 1.1: Let I be an instance of a resource sharing problem. The power of weak preemption for I is $\mathrm{PoWP}(I) = \frac{OPT_{nopmtn}(I)}{OPT_{Wpmtn}(I)}$. The power of strong preemption for I is $\mathrm{PoSP}(I) = \frac{OPT_{nopmtn}(I)}{OPT_{Spmtn}(I)}$.

Definition 1.2: For a class $\mathcal{I}$ of instances, let $\mathrm{PoWP}(\mathcal{I}) = \max_{I\in\mathcal{I}} \mathrm{PoWP}(I)$, and $\mathrm{PoSP}(\mathcal{I}) = \max_{\mathcal{I}\in\mathcal{I}} \mathrm{PoSP}(\mathcal{I})$.

When weak preemptions are allowed, a job may split its processing but must remain in the same copy of a resource. The next definition, of the power of migration, isolates the benefit from allowing jobs to split a task among several different copies of a resources.

Definition 1.3: Let I be an instance of a *Resource Sharing* problem and $\mathcal{I}$ a class of instances. The power of migration for I is $\mathrm{PoM}(I) = \frac{\mathrm{PoWP(I)}}{\mathrm{PoSP(I)}}$, and the power of migration for $\mathcal{I}$ is $\mathrm{PoM}(\mathcal{I}) = \max_{I\in\mathcal{I}} \frac{\mathrm{PoWP}(I)}{\mathrm{PoSP}(I)}$.

We conclude the introduction with an example, demonstrating the power of preemption. Consider an instance I with $n = 6$ jobs and $m = 5$ resource types. The jobs' demands are given in the table in Figure 2. Assume that all resources have the same unit cost per copy. Assume also that $T = 5$. Figure 2 presents two possible schedules. The left schedule is an optimal schedule with weak preemptions. In this schedule exactly one copy of each resource is allocated, therefore $OPT_{Wpmtn}(I) = 5$. The right schedule is an optimal non-preemptive schedule. It holds that $OPT_{nopmtn}(I) = 8$. We can see that the left schedule uses exactly $\left\lceil \sum_j p_{ij}/T \right\rceil$ copies of each resource type i, which is clearly the minimal possible. We conclude that $\mathrm{PoWP}(I) = \mathrm{PoSP}(I) = \frac{8}{5} = 1.6$.

The above example can be generalized to show that for every $n \geq 2$, there exists an instance I_n with n jobs and $m = n+1$ resource types of unit cost such that $\mathrm{PoWP}(I_n) = \mathrm{PoSP}(I_n) = \frac{2n}{n+1} = 2 - \frac{2}{n+1}$. In this construction, one job needs all m resources for a short duration ε, while each of the remaining $n-1$ jobs needs a distinct resource for $T - \varepsilon$ time units. In the non-preemptive schedule, the first and last resources can be shared and require a single copy each, while each of the remaining $m-2$ resources requires two copies, yielding a total cost of $2(n-1)+2 = 2n$. In contrast, a preemptive schedule allows all jobs to be assigned efficiently with only one copy per resource, giving a cost of $m = n+1$, and thus the claimed ratio.

$T=5$	J_1	J_2	J_3	J_4	J_5	J_6
a	1	4	0	0	0	0
b	1	0	4	0	0	0
c	1	0	0	4	0	0
d	1	0	0	0	4	0
e	1	0	0	0	0	4

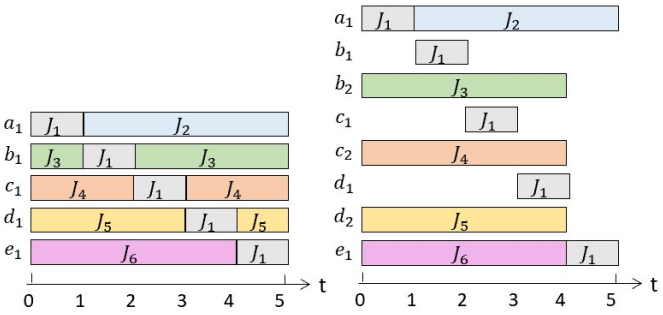

Fig. 2. An example of an instance I for which $\mathrm{PoP}(I) = \frac{8}{5}$.

B. Related Work

There is an extensive body of literature dedicated to job scheduling on parallel machines of various types. As previously mentioned, the resource sharing problem is dual to the well-studied open-shop problem. An instance of *open-shop* scheduling consists of $m \geq 1$ machines (or processors). Each of these machines is capable to perform a single task type. The machines are available at time $t = 0$, and, unlike our model, do not have limited active time, that is, $T = \infty$. In the problem $O||C_{max}$ the goal is to minimize the makespan of the schedule, given by the last completion time of a job. When preemptions are allowed, the problem $O|pmtn|C_{max}$ can be solved in linear-time [6], [10] for any number of machines. In contrast, for the non-preemptive variant, a poly-time algorithm is known only for $m \leq 2$.

For $m \geq 3$, $O||C_{max}$ is NP-complete [6]. A known approximation algorithm guarantees an approximation to the optimal solution that is dependent on the ratio between the maximal total demand for a single machine and the maximal demand of a single job on a single machine [4]. The hardness of the approximation for $O||C_{max}$ has also been studied. It is known that, unless P=NP, there does not exist a polynomial-time approximation algorithm that constructs a schedule with length guaranteed to be strictly less than $\frac{5}{4}$ times the optimal length [17]. When m is fixed, there exists a $(1+\epsilon)$-approximation scheme for the problem that is polynomial in the size of the instance, but exponential in m and $\frac{1}{\epsilon}$ [14]. Optimal algorithm

or better approximal factors were given for restricted classes, such as unit-length jobs [12], [1], or limited number of jobs or machines [3].

The *Power-of-Preemption* (PoP) has been analyzed mainly for the problem $R||C_{max}$ of minimizing the makespan on unrelated parallel machines. In [7] it is shown that the PoP is exactly 4, by showing a lower bound to the PoP, after an upper bound as well as approximation algorithms for restricted classes were given in [2], [13], [15]. The paper [16] studies the PoP and the power of limited preemptions (where only a limited number of preemptions is allowed) for multiple variants of machine environments, job characteristics and objective functions.

Besides the resemblance to the shop-scheduling models, our resource sharing problem with $m = 1$ is identical to the *Bin-Packing* problem (BP). An instance of BP is given by n items, each has a size in (0,1], and the objective is to pack the items into a minimum number of bins, subject to the constraint that the total item sizes in each bin is at most 1. The bin-packing problem has been studied extensively. It is known to be NP-hard and has an FPTAS whose approximation ratio is $(1+\epsilon)$, with a run time of $O(n^{\frac{4}{\epsilon^2}+1})$ [8], [9].

C. Our Results

In this work, we make several contributions that address the scheduling of shared resources under preemptive and non-preemptive settings, and examine how preemptions may reduce the total resource cost. Our results are listed below.

a) Optimal Algorithm with Strong Preemptions: We present a polynomial-time optimal algorithm for the resource sharing problem when strong preemptions are allowed. Our solution is based on a reduction to open-shop scheduling with preemptions ($O|pmtn|C_{\max}$), and reveals that, under strong preemptions, a minimal-cost schedule, in which for all i, there are exactly $\left\lceil \sum_j p_{ij}/T \right\rceil$ copies of resource i, can be computed efficiently.

b) Power of Preemption in the Class $p_{ij} \in \{0, p\}$: For the restricted class of instances in which every task is either of length p or zero, we showed that both non-preemptive and strong-preemptive scheduling can be computed in polynomial time. Yet, the benefit from preemptions may be significant. We prove that the Power of Strong Preemption (PoSP) for this class is bounded by 2, and this is tight.

c) Greedy Heuristics and Empirical Evaluation: Since non-preemptive resource sharing is NP-hard in the general case, we design and evaluate several heuristic algorithms. Our experiments measure how these heuristics perform as problem parameters (number of jobs, resource costs, variation of processing times) vary, and quantify the benefit of preemptions in practice.

Together, these results offer both theoretical and empirical insights into the power of preemptions in shared resource scheduling. We identify tractable cases, establish tight bounds on preemption benefits, and demonstrate through experiments how simple heuristics behave across realistic settings. Our work highlights the fundamental trade-offs between scheduling flexibility, computational complexity, and solution cost.

II. Resource sharing With Strong Preemptions Allowed

In this section we consider settings with strong preemption allowed. We first present and analyze natural greedy algorithms. Similar greedy algorithms are known to be optimal for makespan minimization and for the Bin Packing problem [11]. However, as we show, the greedy approach fails in the resource sharing problem. We then present an optimal, yet more elaborate, polynomial-time optimal algorithm for resource sharing with strong preemptions allowed.

Greedy Algorithm: Algorithm 1 considers the resources one after the other. For every resource i, it assigns the jobs for which $i \in S_j$ one after the other. Every task of length p_{ij} is assigned to the earliest set of disjoint intervals of total length p_{ij} available on open copies of resource i, or on a new copy of resource i if required. The interval must be disjoint with intervals already allocated to previous tasks of job j. Figure 3 presents an execution of Algorithm 1, and demonstrates that it is sub-optimal.

Algorithm 1 - A greedy algorithm for computing a schedule with strong preemptions.

```
All jobs are initially unassigned.
for i = 1 to m do
    for j = 1 to n do
        if i ∈ S_j then: then
            Assign task i of job j on the earliest feasible
            disjoint intervals of total length p_ij on copies of
            resource i. Open a new copy of resource i if
            needed.
        end if
    end for
end for
```

It is easy to see that additional greedy heuristics, which apply a different assignment order (e.g., the outer loop considers the jobs one after the other), are sub-optimal as well,

$T=5$	J_1	J_2	J_3
a	3	3	3
b	2	1	2

Resource demand table

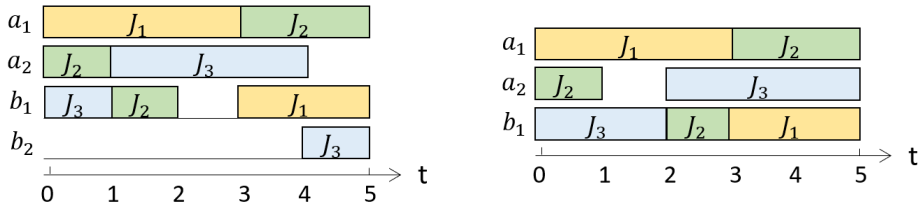

Fig. 3. An example of an instance I with unit-cost resources, demonstrating the suboptimality of a greedy strong-preemptive algorithm. Left: demand table. Center: greedy schedule σ_{alg} with cost 4. Right: optimal schedule $\sigma^\star$ with cost 3.

hinting that an optimal algorithm for resource sharing with strong preemptions, should be more involved.

Optimal Algorithm: Our algorithm for scheduling with strong preemptions allowed employs an optimal algorithm for the problem $O|pmtn|C_{\max}$ as a black box [6]. Recall that in an Open-Shop $O|pmtn|C_{\max}$ problem, we are given a set of n jobs, a set of m machines, and the processing time, p_{ij} of task i of job j. The jobs should be assigned to the machines (a single copy of each machine) such that different tasks of job j are processed in disjoint time intervals. The goal is to minimize the makespan – defined as the last completion time of a job. Let $A_j = \sum_i p_{ij}$ be the total length of job j's tasks. Let $L_i = \sum_j p_{ij}$ be the length of tasks that need to be performed on machine i.

The optimal algorithm for $O|pmtn|C_{\max}$ presented in [6] computes a schedule whose makespan is $\alpha = \max(\max_j A_j, \max_i L_i)$. Since each of these values is a lower bound for the optimal makespan, the algorithm is clearly optimal.

Given an instance of resource sharing, we construct an instance of an open-shop problem as follows:

For every $1 \leq i \leq m$, we define and allocate $\lceil L_i/T \rceil$ machines that will process the tasks of type i. Denote by M_i the set of machines of type i. For every $1 \leq j \leq n$, there is a single job, whose tasks are determined as described in Algorithm 2. The tasks are defined such that for all $i \in S_j$, the total length of tasks on machines in M_i is p_{ij}, and for every machine in M_i, the total length of tasks that need to be processed on M_i is at most T. An example of our reduction is given in Figure 4.

Algorithm 2 - An optimal algorithm with strong preemptions allowed.

1: Given a set J of $n \geq 1$ jobs and a set M of $m \geq 1$ resource types, $p_{ij}, \forall i, j$, and T.
2: For $1 \leq i \leq m$, allocate a set M_i of $\lceil L_i/T \rceil$ machines.
3: Let $m' = \sum_{1 \leq i \leq m} |M_i|$.
4: Construct an instance of $O|pmtn|C_{\max}$ with n jobs and m' machines.
5: **for** $i = 1$ to m **do**
6: **for** $j = 1$ to n **do**
7: **while** $p_{ij} > 0$ **do**
8: Let $i^\star$ be the lowest index of a machine in M_i with total load $l_i^\star < T$.
9: Let $p_{i^\star j} = min(p_{ij}, T - l_i^\star)$.
10: $l_i^\star = l_i^\star + p_{i^\star j}$
11: $p_{ij} = p_{ij} - p_{i^\star j}$
12: **end while**
13: **end for**
14: **end for**
15: Solve $O|pmtn|C_{\max}$ on the resulting instance.

$T=5$	J_1	J_2	J_3
a	3	3	3
b	2	1	2

(a)

	J_1	J_2	J_3
a_1	3	2	0
a_2	0	1	3
b_1	2	1	2

(b)

Fig. 4. An example of our reduction. (a) Resource sharing demand table. (b) Corresponding open-shop instance with $M_a = \{a_1, a_2\}$ and $M_b = \{b_1\}$.

Theorem 2.1: Algorithm 2 computes in polynomial time an optimal solution to the resource sharing problem with strong preemptions allowed.

Proof: Let $\sigma^\star$ be the schedule produced by Algorithm 2. The algorithm reduces the resource sharing instance to an instance of the open-shop problem $O|pmtn|C_{\max}$ by allocating, for each resource i, exactly $\lceil L_i/T \rceil$ machines, and distributing the tasks of each job j across these machines such that the

total processing time per resource equals p_{ij}. This guarantees that the constructed open-shop instance satisfies the time constraints of resource sharing and respects the job-resource requirements.

The schedule $\sigma^\star$ is computed using the optimal algorithm for $O|pmtn|C_{\max}$ from [6], which produces a schedule of makespan $\alpha = \max(\max_j A_j, \max_i B_i)$, where $A_j = \sum_i p_{ij}$ and $B_i = \sum_j p_{ij}$. Our construction guarantees (steps 8-9) that in the open-shop instance, each machine is allocated at most T time units, that is $\forall i, B_i \leq T$. In addition, by definition of the resource rental problem, the total length of each job is at most T, that is, $\forall j, A_j \leq T$. This implies that $\alpha \leq T$, and the resulting makespan is at most T.

Since each machine in the open-shop solution corresponds to one copy of a resource in resource sharing, and all jobs are scheduled such that their tasks are disjoint on each machine, the resulting schedule is feasible in resource sharing under strong preemptions. Furthermore, the number of machines used for each resource is the minimal possible given the total load and capacity T, meaning the total cost (i.e., number of machines) is minimized. Hence, $\sigma^\star$ is a feasible resource sharing schedule with minimal cost. The time complexity of the algorithm is clearly polynomial and is dominated by the time complexity of the algorithm for $O|pmtn|C_{\max}$ [6], which is $\mathcal{O}(r^2)$, where r is the number of non-zero tasks - typically smaller than $\mathcal{O}(n^2)$. ∎

III. The Class $\mathcal{I}_{0,p}$: instances in which $p_{ij} \in \{0, p\}$

In this section we analyze the class $\mathcal{I}_{0,p}$ of resource sharing instances with $p_{ij} \in \{0, p\}$. That is, all the non-empty tasks have the same length p. As a restricted class of the general problem, our optimal algorithm for strong-preemptive applies for this class as well, making the strong-preemptive problem polynomially solvable.

We show that this class is polynomially solvable even when preemptions are not allowed. Given an instance in $\mathcal{I}_{0,p}$, let r_i denote the number of the non-zero tasks which require resource i. Thus, the total demand for resource i is $p \cdot r_i$. Our optimal solution reduces the problem to $O|p_{ij} \in \{0,1\}|C_{\max}$, that is, minimum makespan in an open-shop environment with unit processing times. An optimal poly-time solution for this problem, based on a reduction to minimum edge coloring in a bipartite graph, is presented in [17].

Overview of our algorithm: In the first step, we determine how many copies are required from each resource type. It is easy to see that for all $1 \leq i \leq m$, $\lceil r_i / \lfloor T/p \rfloor \rceil$ is a lower bound for this number. We define an instance of $O|p_{ij} \in \{0,1\}|C_{\max}$ with $\lceil r_i/\lfloor T/p \rfloor \rceil$ machines of type i. We then use the algorithm in [17] to get an optimal schedule for the resulting $O|p_{ij} \in \{0,1\}|C_{\max}$ problem, and conclude an optimal solution to our resource rental problem. Formally:

Theorem 3.1: Computing an optimal solution for a resource sharing problem in $\mathcal{I}_{0,p}$ can be done in polynomial time.

Proof: Let I be an instance of resource sharing in $\mathcal{I}_{0,p}$, with availability time T. For each resource i, let r_i denote the number of jobs j for which $p_{ij} = p$. Since each job requires either 0 or p units on each resource, and each copy of i can process at most $\lfloor T/p \rfloor$ tasks of length p, it follows that at least $\lceil r_i/\lfloor T/p \rfloor \rceil$ copies of resource i are required.

We reduce this instance to an instance of open-shop scheduling $O|p_{ij} \in \{0,1\}|C_{\max}$ with unit-length tasks. Note that since $p_{ij} \in \{0,1\}$, such an instance is defined by determining a set of machines, a set of jobs, and the subset of machines required by each job. For each resource i in the resource rental problem, we introduce $\lceil r_i/\lfloor T/p \rfloor \rceil$ machines of type i, and for every job j with $p_{ij} = p$, we add one of these machines to the subset of machines required by job j. This assignment is done such that each machine is required by at most $\lfloor T/p \rfloor$ jobs. In this way, we guarantee that the load on every machine in the resulting open-shop instance is at most $\lfloor T/p \rfloor$.

The constructed instance is a valid $O|p_{ij} \in \{0,1\}|C_{\max}$ problem, which is known to admit a polynomial-time optimal solution via a reduction to minimum edge coloring in bipartite graphs [17]. The solution is the maximum between the load on a single machine and the total demand of a job. Our construction guarantees that this value is at most $\lfloor T/p \rfloor$. The resulting open-shop schedule induces a feasible resource sharing schedule by stretching each unit-length task to a length p. The resulting makespan is T - that fits the availability time of the resources.

Since our construction uses the minimal number of machines per resource (matching the lower bound), and produces a feasible schedule, it is optimal. We conclude that resource sharing with $p_{ij} \in \{0, p\}$ can be solved optimally in polynomial time. ∎

Next, we bound the power of strong preemption for this class. It is interesting to note that $PoSP > 1$ even though the problem is polynomially solvable in both the preemptive and the non-preemptive models. Thus, preemptions do not affect the computational complexity of the problem but reduce the cost of an optimal solution.

Theorem 3.2: $\text{PoSP}(\mathcal{I}_{0,p}) = 2$.

Proof: Let $I \in \mathcal{I}_{0,p}$ be a resource sharing instance with $p_{ij} \in \{0, p\}$, time T, and resource costs c_i. Let r_i denote

the number of jobs that require resource i. The total load on resource i is therefore $r_i \cdot p$.

As discussed in Section II, the cost of an optimal solution with strong preemptions allowed is

$$\text{cost}_{\text{pre}}(I) = \sum_i \left\lceil \frac{\sum_j p_{ij}}{T} \right\rceil c_i = \sum_i \left\lceil \frac{r_i \cdot p}{T} \right\rceil c_i = \sum_i \left\lceil \frac{r_i}{T/p} \right\rceil c_i,$$

and the cost of an optimal solution *without* preemptions is:

$$\text{cost}_{\text{non-pre}}(I) = \sum_i \left\lceil \frac{r_i}{\lfloor T/p \rfloor} \right\rceil c_i.$$

We first note that if p divides T, then an optimal solution with preemptions allowed, does not use any preemptions, implying that $PoSP(I) = 1$. If p does not divide T, then since $p \leq T$, it holds that $\frac{T}{p} \geq 1$, implying $(T/p)/\lfloor T/p \rfloor < 2$. Therefore,

$$PoSP(I) = \frac{\sum_i \lceil r_i / \lfloor \frac{T}{p} \rfloor \rceil c_i}{\sum_i \left\lceil r_i / \frac{T}{p} \right\rceil c_i} < 2.$$

We show that the above analysis is tight, that is, we can get arbitrarily close to a $PoSP$ of 2. Specifically, given $0 < \delta \leq \frac{1}{2}$, we present an instance I such that $PoSP(I) \geq 2 - \delta$. As we show, a tight example exists already for instances with a single resource type.

Given $0 < \delta \leq \frac{1}{2}$, let $\epsilon \leq \delta$ such that $\frac{2-\epsilon}{\epsilon}$ and $\frac{1}{\epsilon}$ are integers. Let $T = 1$, $m = 1$, $n = \frac{2-\epsilon}{\epsilon}$, $p = \frac{1}{2-\epsilon}$ and $p_{1,j} = p$ for any $1 \leq j \leq n$.

Since $\frac{1}{2} < p \leq 1$, without preemptions no two jobs can share a copy of the resource, therefore, n copies of resource i are required, implying that the cost on any solution without preemptions is exactly $n = \frac{2-\epsilon}{\epsilon} = \frac{2}{\epsilon} - 1$.

On the other hand, the number of copies required with preemptions allowed is

$$\left\lceil \frac{r_1}{\frac{T}{p}} \right\rceil = \left\lceil \frac{n}{\frac{T}{p}} \right\rceil = \left\lceil \frac{(2-\epsilon)/\epsilon}{1/\frac{1}{2-\epsilon}} \right\rceil = \left\lceil \frac{(2-\epsilon)/\epsilon}{2-\epsilon} \right\rceil = \left\lceil \frac{1}{\epsilon} \right\rceil = \frac{1}{\epsilon}.$$

Therefore,

$$PoSP(I) = \frac{2/\epsilon - 1}{1/\epsilon} = \frac{2/\epsilon}{1/\epsilon} - \frac{1}{1/\epsilon} = 2 - \epsilon \geq 2 - \delta.$$

■

IV. Greedy Heuristics and Empirical Evaluation Experiment

A. Key Questions

In our experiments, we addressed the following questions regarding the performance of some natural greedy heuristics and the impact of preemptions in the resource sharing setting:

1) **Evaluate the performance of natural greedy algorithms**
   Since non-preemptive resource sharing is NP-hard, we rely on greedy heuristics as approximate solutions. Our goal is to test whether even simple heuristics produce significant cut down in resource costs.
2) **Investigate how parameter variations influence the efficiency of allocations with no preemptions, weak, and strong preemptions.**
   We systematically vary the number of jobs (n), the number of resources (m) and the variance of distributions of job lengths (p_{ij}). Our goal is to see how these changes amplify or diminish the empirical advantages of preemptions compared to non-preemptive allocations.
3) **Estimating the Power of Preemption.** For the restricted class $p_{ij} \in \{0, p\}$, we proved that PoSP cannot exceed 2 (see Section III). For general instances, we lack a theoretical upper bound for either PoSP or PoWP. To explore them, we compare the performance of our three algorithms (non-preemptive, weak-preemptive, and strong-preemptive). Even while not optimal, these heuristics provide a practical view of preemption benefits across diverse instances.

B. Experiment Design

In order to address the key questions outlined in Section IV-A, we built a simulator that generates a variety of resource sharing instances and evaluated the schedules produced for these instances by several greedy heuristics under different preemptive settings.

We measured the performance of each heuristic by comparing the total cost of its resulting schedule against two baseline values:

- The **optimal cost**, achieved when strong preemptions are allowed, and given by
$$C_{\text{opt}}(I) = \sum_{i \in M} c_i \cdot \left\lceil \frac{\sum_{j \in \mathcal{J}} p_{ij}}{T} \right\rceil.$$
- The **no-sharing cost**, which assumes that each non-empty task $p_{ij} > 0$ is assigned its own dedicated copy. This cost represents a naive allocation strategy and is given by:
$$C_{\text{no-share}}(I) = \sum_{i \in M} c_i \cdot |\{j \in \mathcal{J} \mid p_{ij} > 0\}|.$$

When evaluating the efficiency of various heuristics, we used the no-sharing cost as a normalization baseline.

For simplicity, all instances used in our experiments assume unit resource costs; that is, $c_i = 1$ for all $i \in M$. This

allows us to focus solely on the structural differences between scheduling strategies without the added complexity of cost heterogeneity.

1) Heuristics: We implemented three heuristics to resource sharing, corresponding to non-preemptive, weak-preemptive, and strong-preemptive scheduling. All three heuristics share the same core idea: iteratively assign each job's tasks to available resource copies using a first-fit strategy.

For scheduling with strong-preemptions we have used Algorithm 1, augmented with the following resource-sorting step: Before assigning a job, we sort its required resources in descending order of task duration. That is, for each job j, we reorder its set of required resources S_j according to p_{ij}, scheduling longer tasks first. This helps accommodating shorter tasks in the remaining gaps, and by that improving success rate of sharing the resources. For scheduling with weak-preemptions we used the same algorithm; however, in Step 5, we only considered assignments on a single copy.

Algorithm 3 below describes our heuristic for non-preemptive scheduling. In the non-preemptive setting, partial assignments may block later tasks of the same job and prevent any feasible solution. For example, if $T = 6$ and a job consists of two tasks of length 3, then the second task cannot be performed non-preemptively if the first task is not processed in $[0, 3)$ or in $[3, 6)$. To address this, we keep for each job a variable $left_j$ that stores the leftmost time-point in $[0, T]$ in which it is assigned. We schedule the job using a left-aligned strategy: tasks are scheduled one after the other, with each task placed as early as possible (starting at a leftmost time) and within a carefully restricted interval to ensure that it ends early enough -enabling the remaining tasks to be assigned such that the job completes by time T. If no such interval exists on an already-open copy of the corresponding resource, then a new copy is activated.

All three algorithms share the same Python implementation framework, managing available time segments in each machine's schedule and placing tasks accordingly. The final cost is computed by summing up the number of machines used for each resource.

2) Dataset Generation: Our method for creating benchmark instances generates task durations for each job in a consistent, yet randomized manner. Every instance is characterized by four parameters: number of jobs, n; number of resources, m; task-length variance $0 \leq \texttt{variance} \leq 1$; and job length as a fraction of T, $\texttt{span} \leq 1$.

For a given set of parameters, we construct a set of n jobs each consisting of m tasks, such that (i) for each job $1 \leq j \leq n$ and resource $1 \leq i \leq m$, with probability 0.2 the task of job j corresponding to resource i is empty, that is, $p_{ij} = 0$. (ii) Each of the remaining tasks has a length drawn uniformly from the range $[1-\texttt{variance}, 1+\texttt{variance}]$, and then scaled such that the total tasks length is $\texttt{span}$. We assume that $T = 1$. This is w.l.o.g. since in the last step, the task durations are scaled such that the total job length is $\texttt{span} \cdot T$.

Algorithm 3 Greedy Algorithms for Non-Preemptive Scheduling

1: **for** each job $j \in \mathcal{J}$ **do**
2: Sort S_j (the required resources of j) by non-increasing order of p_{ij}
3: Let $left_j \leftarrow 0$, $p_j \leftarrow \sum_{i \in S_j} p_{ij}$, $scheduled \leftarrow 0$
4: **for** each $i \in S_j$ where $p_{ij} > 0$ **do**
5: Let $remaining \leftarrow p_j - scheduled$
6: Let $latest\_end \leftarrow T - (remaining - p_{ij})$
7: Assign task (i, j) as a single interval of length p_{ij} on an existing copy, or a new copy of resource i within $[left_j, latest\_end]$
8: Let $left_j \leftarrow$ end time of assigned interval
9: Let $scheduled \leftarrow scheduled + p_{ij}$
10: Update current solution and resource state
11: **end for**
12: **end for**

To evaluate how different structural parameters affect scheduling performance, we generated four large sets of instances, each containing 1000 samples. In each set, we fixed three parameters and systematically varied the fourth: number of jobs, number of resources, variance, or span. This design isolates the influence of each factor on the heuristics' relative performance. When fixed, we used the following parameters: $n = 10$ jobs, $m = 3$ resources, variance 0.9, and span 1. Our experiments show that these values best capture the characteristics of each heuristic.

V. Experimental Results

We begin by presenting the raw results of our experiments. Figures 5- 8 summarize the efficiency of each of the heuristics with respect to four parameters: number of jobs, number of resources, variance, and span. For each experiment (of 1000 samples) we calculated the average solution cost of the three heuristics, as well as the optimal cost. These costs are normalized relative to the no-sharing baseline, giving the *efficiency* of each heuristic, which is depicted in the figures.

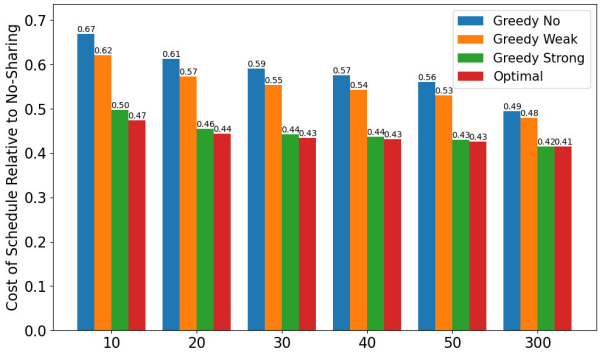

Fig. 5. Efficiency for variable the number of jobs.

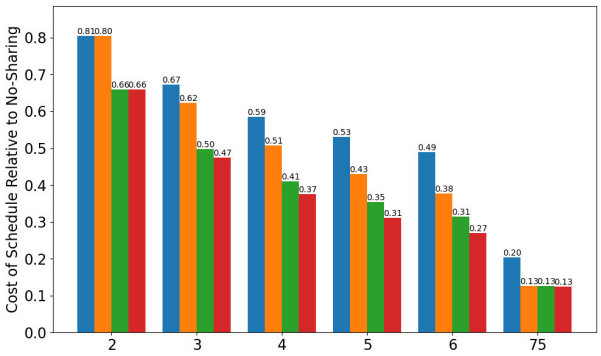

Fig. 6. Efficiency for variable number of resources.

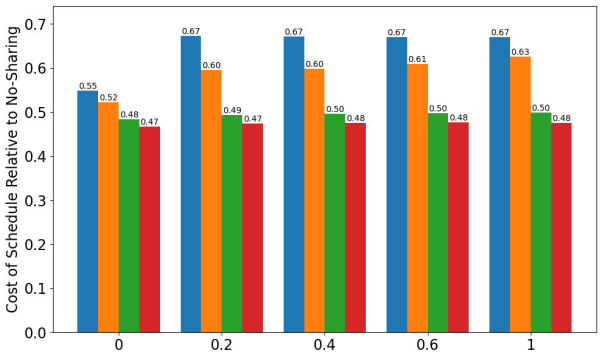

Fig. 7. Efficiency for variable `variance`.

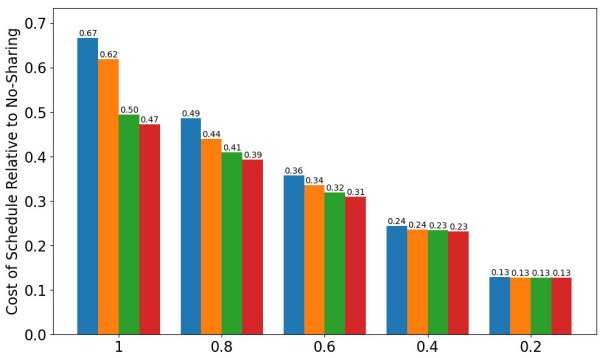

Fig. 8. Efficiency for variable `span` (total job length as a fraction of T).

We now turn to interpreting these results in light of the three research questions introduced earlier.

A. (Q1) How well do the greedy algorithms approximate optimal schedules?

Our results show that all greedy algorithms remain within a constant factor of the optimal strong-preemptive schedule across all tested parameters. In favorable cases, greedy schedules approximate the optimum within ratio 1.1. In more constrained scenarios (e.g., with many resources), the gap may increase, but none of the heuristics exceeded a factor of 2. This empirical observation supports the practical use of these heuristics even in the absence of theoretical guarantees.

B. (Q2) What is the effect of parameter variations on preemption benefits?

We observed how each structural parameter shaped the cost gap between non-preemptive, weak-preemptive, and strong-preemptive settings:

a) Number of Jobs (n): As n increases, all algorithms achieve improved efficiency relative to the no-sharing baseline. The strong-preemptive heuristic continues to closely approximate the optimal cost. Meanwhile, the non-preemptive heuristic gets increasingly closer to the weak-preemptive approach, and both converge to approximately 1.25 times the cost of the optimal solution.

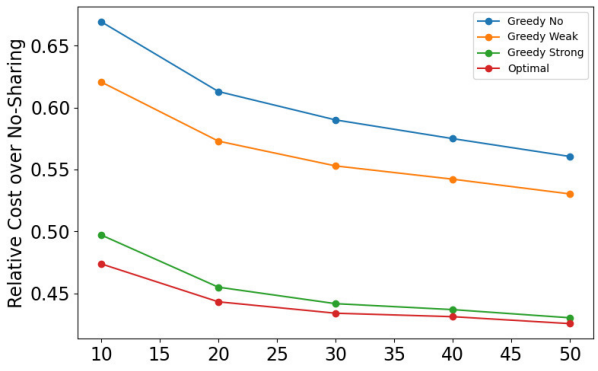

Fig. 9. Relative cost over no-sharing cost as number of jobs increases.

b) Number of Resources (m): As m increases, all of our algorithms achieve improved efficiency relative to the no-sharing baseline. The strong-preemptive heuristic continues to produce near-optimal results, and the weak-preemptive algorithm closely follows it, with the gap between them narrowing as m grows. The non-preemptive heuristic also benefits from increased resources, but to a lesser extent - its relative cost remains around 1.5 times the optimal.

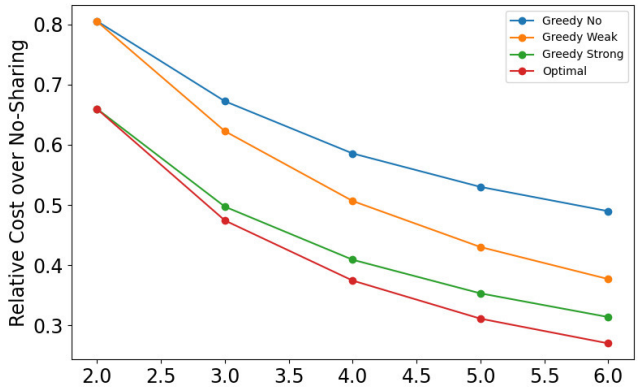

Fig. 10. Relative cost over no-sharing cost as number of resources increases.

c) Variance of Task Durations: Our results indicate that variance in task durations significantly impacts scheduling performance when it first appears (transitioning from zero variance to some positive value). However, beyond this initial effect, the magnitude of variance has minimal influence on the algorithm performance. Once some level of variance exists, increasing it further does not substantially change the gap between the heuristics, suggesting that it is the presence of variance, rather than its magnitude, that defines the difficulty of the instances.

d) Span: Recall that the span determines the total length of the jobs' tasks. As expected, decreasing the span consistently makes the problem easier, since tasks can be more flexibly scheduled without conflicts, leading to lower overall costs. Our results show that beyond a certain point, the problem becomes nearly trivial, with all algorithms, in particular the non-preemptive one, achieving near-optimal results.

C. (Q3) Estimating the Power of Preemption in General Cases

In the general case, across a wide range of tested instances with relatively low values of n and m, the three approaches—non-preemptive, weak-preemptive, and strong-preemptive - are relatively comparable. In those values, we find that non-preemptive schedules cost on average approximately 50% more than strong-preemptive ones, while weak-preemptive schedules cost about 25% more than strong-preemptive schedules. Thus, on average, the *power of strong preemption* is $PoSP(I) = 1.5$, and the *power of migration* is $PoM(I) = 1.25$. While these results are based on smaller parameter settings, the trends observed in earlier sections indicate similar or even stronger effects for larger instances. In particular, as the number of jobs increases, the non-preemptive and weak-preemptive heuristics converges toward a relative cost of approximately 1.25 compared to the optimal. While as the number of resources increases, the weak-preemptive algorithm achieves near-optimal performance, closing the gap with the strong-preemptive approach.

VI. Summary and Future Work

In this paper we analyzed a scheduling problem arising in environments in which resources may be shared by several users. The goal is to minimize the total cost of allocated resources. We summarize below our contribution and the main takeaways from our theoretical and empirical study of the resource sharing problem.

a) Optimal Algorithm for Scheduling with Preemptions Allowed: We developed an algorithm for solving the problem optimally when strong preemptions are allowed. Our optimal algorithm is suitable for all classes of instances, and achieves the lowest possible cost, given by

$$C_{\text{opt}}(I) = \sum_{i \in M} c_i \cdot \left\lceil \frac{\sum_{j \in \mathcal{J}} p_{ij}}{T} \right\rceil .$$

b) Tight analysis of the class $\mathcal{I}_{0,p}$: For the class $\mathcal{I}_{0,p}$, we developed an optimal algorithm for scheduling without preemptions and established a tight bound of 2 for the power of strong preemptions. We conjecture that this bound is valid for additional classes.

c) Greedy Approaches and Approximation: We developed three greedy heuristics for resource sharing: non-preemptive, weak-preemptive, and strong-preemptive, and found that all provide efficient approximations. As expected, the weak and strong preemptive variants consistently outperform the non-preemptive approach. In all parameter regimes, the heuristics remained within a constant factor of the optimal cost.

Our work leaves open several directions for future work:

1) *Theoretical analysis of the general case:* A deeper understanding of the general case is still needed. Two specific directions stand out:
   - Determining tight upper bounds for the power of weak and strong preemptions (PoWP and PoSP), as well as the power of migration (PoM).
   - Deriving theoretical worst-case approximation guarantees for the proposed heuristics, complementing the observed empirical performance.
2) *Analysis of additional restricted classes of instances:* Beyond the $\mathcal{I}_{0,p}$ class, it would be interesting to analyze additional classes of instances, arising in real-world applications.

- Instances with a constant number of job types, where all jobs of the same type have the same resource requirements.
- Instances with job-dependent or resource-dependent processing times, e.g., $p_{ij} = p_j$ or $p_{ij} = p_i$.
- Instances in which each job consists of a bounded number of tasks (e.g., $|S_j| = 2$ for all j).

For each such class, future work should aim to determine whether an optimal solution can be computed in polynomial time, or alternatively prove hardness and develop approximation algorithms.

3) *Modeling costs for preemptions and migrations:* Our model assumes that preemptions and migrations are not associated with a cost. In real-world settings, however, splitting or migrating tasks may involve setup or transition costs. Extending the model to incorporate these costs may yield richer structural insights and lead to new trade-offs between flexibility and efficiency. Analyzing the effect of such costs on both the complexity and the power of preemptions is a promising direction for future work.

REFERENCES

[1] H. Bräsel, D. Kluge, and F. Werner, A polynomial algorithm for the $[n/m/0, t_ij = 1, tree/C_{max}]$ open shop problem, European Journal of Operational Research, 72(1):125–134, 1994.

[2] R. Canetti and S. Irani. Bounding the power of preemption in randomized scheduling. *SIAM Journal on Computing*, 27(4):993–1015, 1998.

[3] I. G. Drobouchevitch. Three-machine open shop with a bottleneck machine revisited. *Journal of Scheduling*, 24(2):197–208, 2021.

[4] T. Fiala. An algorithm for the open-shop problem. *Mathematics of Operations Research*, 8(1):100–109, 1983.

[5] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, 1979.

[6] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *J. ACM*, 23(4):665–679, 1976.

[7] J.R.Correa, M.Skutella, and J.Verschae. The power of preemption on unrelated machines and applications to scheduling orders. *Mathematics of Operations Research*, 37(2):379–398, 2012.

[8] N. Karmakar and R. Karp. An efficient approximation scheme for the one-dimensional bin packing problem. In *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, pages 312–320, 1982.

[9] A. Levin. Approximation schemes for the generalized extensible bin packing problem. *Algorithmica*, 84(2):325–343, 2022.

[10] C. Liu and R. Bulfin. Scheduling open shops with unit execution times to minimize functions of due dates. *Operations Research*, 36(4):553–559, 1988.

[11] R. McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6:1–12, 1959.

[12] B. Naderi, M. Zandieh, and M. Yazdani. Polynomial time approximation algorithms for proportionate open-shop scheduling. *International Transactions in Operational Research*, 21(6):1031–1044, 2014.

[13] A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002.

[14] S. Sevastyanov and G. Woeginger. Makespan minimization in open shops: A polynomial time approximation scheme. *Math. Program.*, 82:191–198, 1998.

[15] H. Shachnai and T. Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, 32(4):651–678, 2002.

[16] B. Takand. Towards power of preemption on parallel machines. *MPhil thesis, University of Greenwich*, 2016.

[17] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'janov, and D. B. Shmoys. Short shop schedules. *Operations Research*, 45(2):288–294, 1997.

[18] R. Zhang, N. Horesh, E. Kontou, and Y, Zhou, Electric vehicle community charging hubs in multi-unit dwellings: Scheduling and techno-economic assessment, *Transportation Research Part D: Transport and Environment*, Vol. 120, 2023.