Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Scheduling with Complete Multipartite Incompatibility Graph on Parallel Machines

Tytus Pikies,[1] **Krzysztof Turowski,** [2] **Marek Kubale** [1]

[1] Department of Algorithms and System Modeling, Gdańsk University of Technology, 80-233 Gdańsk, Poland
[2] Theoretical Computer Science Department, Jagiellonian University, 30-348 Kraków, Poland
tytpikie@pg.edu.pl, krzysztof.szymon.turowski@gmail.com, kubale@eti.pg.edu.pl

Abstract

In this paper we consider a problem of job scheduling on parallel machines with a presence of incompatibilities between jobs. The incompatibility relation can be modeled as a complete multipartite graph in which each edge denotes a pair of jobs that cannot be scheduled on the same machine.

We provide several results concerning schedules, optimal or approximate with respect to the two most popular criteria of optimality: $C_{\max}$ (makespan) and $\sum C_j$ (total completion time). We consider a variety of machine types in our paper: identical, uniform, and unrelated. Our results consist of delimitation of the easy (polynomial) and NP-hard problems within these constraints. We also provide algorithms, either polynomial exact algorithms for the easier problems, or algorithms with a guaranteed constant worst-case approximation ratio.

In particular, we fill the gap on research for the problem of finding a schedule with the smallest $\sum C_j$ on uniform machines. We address this problem by developing a linear programming relaxation technique with an appropriate rounding, which to our knowledge is a novelty for this criterion in the considered setting.

Introduction

Imagine that we are treating some people ill with contagious diseases. There are quarantine units containing people ill with a particular disease waiting to receive some medical services. We also have a set of nurses. We would like the nurses to perform the services in a way that no nurse will travel between different quarantine units, to avoid spreading of the diseases. Also, we would like to provide each patient with the required services, which correspond to the time to be spent by a nurse.

Consider two sample goals: The first might be to lift the quarantine in the general as fast as possible. The second might be to minimize the average time of patient treatment.

The problem can be easily modeled as a scheduling problem in our setting. The jobs are the medical services to be performed. The division of jobs into parts of the incompatibility graph is the division of the tasks into the quarantine units. The machines are the nurses. The sample goals correspond to $C_{\max}$ and $\sum C_j$ criteria, respectively.

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

This is merely a single example of an application of *scheduling with incompatibility graph on parallel machines*.

Notation and Description of the Problems

We follow the notation and definitions from Brucker (1999). Let the set of jobs be $J = \{j_1, \ldots, j_n\}$ and the set of machines be $M = \{m_1, \ldots, m_m\}$. We denote the processing requirements of $j_1, \ldots, j_n$ as $p_1, \ldots, p_n$, respectively.

Now let us define a function $p : J \times M \to \mathbb{N}$, which assigns a time needed to process a given job for a given machine. We distinguish three main types of machines, in the ascending order of generality:

- *identical* – when $p(j_i, m_l) = p_i$ for all $j_i \in J$, $m_l \in M$,
- *uniform* – when there exists a function $s : M \to \mathbb{Q}_+$, such that $p(j_i, m_l) = \frac{p_i}{s(m_l)}$ for any $j_i \in J$, $m_l \in M$,
- *unrelated* – when there exists $s : J \times M \to \mathbb{Q}_+$, such that $p(j_i, m_l) = \frac{p_i}{s(j_i, m_l)}$, for any $j_i \in J$, $m_l \in M$.

The incompatibility between jobs is a relation that can be represented as a simple graph $G = (J, E)$, where J is the set of jobs, and $\{j_1, j_2\}$ belongs to E, iff j_1 and j_2 are incompatible. In this paper we consider complete multipartite graphs, i.e., graphs whose sets of vertices may be split into disjoint independent sets $J_1, \ldots, J_k$ (called *parts* of the graph), such that for every two vertices in different parts there is an edge between them. Due to the fact that the structure is simple, we identify the graph with the partition of the jobs.

We differentiate between the cases when the number of the parts is fixed, and when this is not the case. In the former case we denote the graph as $G =$ *complete k-partite*, and in the latter as $G =$ *complete multipartite*.

A schedule S is an assignment from jobs to the set of machines and starting times. Hence if $S(j) = (m_l, t)$, then job j is executed on machine m_l in time interval $[t, t + p(j, m_l))$ and $t + p(j, m_l) = C_j$ is completion time of j in S. No two jobs may be executed at the same time on any machine. Moreover, no two jobs which are connected by an edge in the incompatibility graph may be scheduled on the same machine. By $C_{\max}(S)$ we denote maximum C_j in S over all jobs. By $\sum C_j(S)$ we denote a sum of completion times of jobs in S. These are two criteria of optimality of a schedule commonly considered in the literature. Note that in both cases we are interested in the minimization of the respective measure.

We recall an observation that a jobs-to-machines assignment is sufficient to determine the optimal values of these measures. Clearly $C_{\max}$ is indifferent to permutations of the tasks on the same machine and for $\sum C_j$ it is well known that the best permutation on any machine is given by Smith's Rule (Smith 1956), i.e., according to non-decreasing processing requirements.

Throughout the paper we use the well-known notation $\alpha|\beta|\gamma$ of Lawler, Lenstra, and Kan (1982). In particular, we are interested in problems where:

- α is either P (identical machines), or Q (uniform machines), or R (unrelated machines),
- β contains either $G =$ *complete multipartite* or $G =$ *complete k-partite*, and some additional constraints, e.g., $p_j = 1$ (unit jobs only),
- γ is either $C_{\max}$ or $\sum C_j$.

An Overview of the Previous Work

We recall that the $P||C_{\max}$ is NP-hard even for two machines (Garey and Johnson 1979). However, $Q||C_{\max}$ (and therefore $P||C_{\max}$ as well) does admit a PTAS (Hochbaum and Shmoys 1988). Moreover, $R_m||C_{\max}$ admits an FPTAS (Horowitz and Sahni 1976). There is a $(2 - \frac{1}{m})$-approximation algorithm for $R||C_{\max}$ (Shchepin and Vakhania 2005); however there is no polynomial algorithm with approximation ratio better than $\frac{3}{2}$, unless P = NP (Lenstra, Shmoys, and Tardos 1990). On the other hand, $Q|p_j = 1|C_{\max}$ and $Q||\sum C_j$, can be solved in $O(\min\{n + m \log m, n \log m\})$ (Dessouky et al. 1990) and $O(n \log n)$ (Brucker 1999, p. 133–134) time, respectively. Moreover, $R||\sum C_j$ can be regarded as a special case of an assignment problem (Bruno, Coffman Jr, and Sethi 1974), which can be solved in polynomial time.

The problem of scheduling with incompatible jobs for identical machines was introduced in (Bodlaender, Jansen, and Woeginger 1994). They provided a series of polynomial time approximation algorithms for $P|G =$ *k-colorable*$|C_{\max}$. For bipartite graphs they showed that $P|G =$ *bipartite*$|C_{\max}$ has a polynomial 2-approximation algorithm, and this ratio of approximation is the best possible if P ≠ NP. They also proved that there exists an FPTAS in the case when the number of machines is fixed and G has constant treewidth.

The special case $P|G, p_j = 1|C_{\max}$ was treated extensively in the literature under the name Bounded Independent Sets: for given m and t, determine whether G can be partitioned into at most t independent sets with at most m vertices in each. More generally, $P|G|C_{\max}$ is equivalent to a weighted version of Bounded Independent Sets. We note also that $P|G, p_j = 1|C_{\max}$ is closely tied to Mutual Exclusion Scheduling, where we are looking for a schedule in which no two jobs which are connected by an edge in G are executed at the same time. For the unrestricted number of machines it is the case that $P|G, p_j = 1|C_{\max}$ has a polynomial algorithm for a certain class of graphs G if and only if Mutual Exclusion Scheduling has a polynomial algorithm for the same class of graphs. When all this is taken into account, there are known polynomial algorithms for solving $P|G, p_j = 1|C_{\max}$ when G is restricted to the following classes: forests (Baker and Coffman Jr 1996), split graphs (Lonc 1991), complements of bipartite graphs and complements of interval graphs (Bodlaender and Jansen 1995). However, the problem remains NP-hard when G is restricted to bipartite graphs (even for 3 machines), interval graphs and cographs (Bodlaender and Jansen 1995).

Recently another line of research was established for G equal to a union of cliques (bags) by Das and Wiese (2017). The authors considered $C_{\max}$ criterion and presented a PTAS for identical machines together with $(\log n)^{1/4-\epsilon}$-inapproximability result for unrelated machines.

They also provided an 8-approximate algorithm for unrelated machines with additional constraints. This approach was further pursued in Grage, Jansen, and Klein (2019), where an EPTAS for identical machines case was presented. The last result is a construction of a PTAS for uniform machines with some additional restrictions on machine speeds and bag sizes (Page and Solis-Oba 2020). For the definition of PTAS and other approximation schemes see e.g., Epstein and Sgall (2004), Kones and Levin (2019), or Jansen and Maack (2019).

Unfortunately, the case of the complete multipartite incompatibility graph was not studied so extensively. It may be inferred from Bodlaender, Jansen, and Woeginger (1994) that for $P|G =$ *complete multipartite*$|C_{\max}$ there exists a PTAS, which can be easily extended to EPTAS; and that there is a polynomial time algorithm for $P|G =$ *complete multipartite*$, p_j = 1|C_{\max}$.

When G is complete multipartite, it has to be the case that each machine serves jobs only from one part of the graph. Therefore, it is a special case of the model, where each machine may serve jobs from c different parts. This model was investigated by Jansen, Lassota, and Maack (2020), and it follows from their work that there exists a PTAS for $P|G =$ *complete multipartite*$|C_{\max}$ even in this more general setting.

In the case of uniform machines Mallek, Bendraouche, and Boudhar (2019) proved that $Q|G =$ *complete 2-partite*$, p_j = 1|C_{\max}$ is NP-hard, but it may be solved in $O(n)$ time when the number of machines is fixed. Moreover, they showed an $O(mn + m^2 \log m)$ algorithm for the particular case $Q|G = star, p_j = 1|C_{\max}$. However, their result implicitly assumed that the number of jobs n is not encoded – as it is customary assumed – in unary form, but in binary on $\log n$ bits thus making the size of output schedules exponential in terms of the input size.

Our Results

In this paper we provide several results for different combinations of machines types (identical, uniform, or unrelated), graphs (complete multipartite with a number of parts as a problem parameter or as an input), and optimality criterion ($C_{\max}$ or $\sum C_j$).

We summarize our results in Table 1. We grouped results for each types of machine, and then for $C_{\max}$ and $\sum C_j$.

$P\|G = \textit{complete multipartite}\|C_{\max}$	NP-hard		(Garey and Johnson 1979)
	EPTAS	polynomial time	(Jansen et al. 2020)
$P\|G = \textit{complete multipartite}\|\sum C_j$	exact	$\mathrm{O}(mn + n\log n)$	Corollary 1
$Q\|G = \textit{complete multipartite}, p_j = 1\|C_{\max}$	Strongly NP-hard		Theorem 2
	2-approximation	$\mathrm{O}(mn\log(mn))$	Theorem 3
$Q\|G = \textit{complete k-partite}, p_j = 1\|C_{\max}$	exact	$\mathrm{O}(mn^{k+1}\log(mn))$	Theorem 5
$Q\|G = \textit{complete multipartite}, p_j = 1\|\sum C_j$	Strongly NP-hard		Theorem 1
	4-approximation	$\mathrm{O}(m^2n^3\log m)$	Theorem 4
$Q\|G = \textit{complete k-partite}, p_j = 1\|\sum C_j$	exact	$\mathrm{O}(mn^{k+1})$	Theorem 6
$Q\|G = \textit{complete k-partite}\|\sum C_j$	4-approximation	polynomial time	Theorem 7
$R\|G = \textit{complete 2-partite}, p_j \in \{a,b\}\|C_{\max}$	no $\mathrm{O}(1)$ approximation in polynomial time		Theorem 8
$R\|G = \textit{complete 2-partite}, p_j \in \{a,b\}\|\sum C_j$	no $\mathrm{O}(1)$ approximation in polynomial time		Theorem 8

Table 1: Summary of the results proved in this paper.

Identical Machines

We recall that $P|G = \textit{complete multipartite}|C_{\max}$ is a generalization of $P||C_{\max}$ (because *empty* = *complete 1-partite*), hence it is also Strongly NP-hard. However, it admits an EPTAS, as can be inferred from (Bodlaender, Jansen, and Woeginger 1994). In this section we prove that there exists an algorithm with polynomial running time for the same problem, but with another criterion, namely $P|G = \textit{complete multipartite}|\sum C_j$. It turns out that a greedy approach is sufficient to solve the problem.

Let us define what we mean by a *greedy assignment* of machines to parts:

1. assign to each part a single machine,
2. assign the remaining machines one by one to the parts in a way that it decreases $\sum C_j$ in this step as much as possible.

To see why this approach works we need the following lemma, which proves non-increasing gains from assigning consecutive machines to any single part (i.e., to an empty subgraph of G).

Lemma 1. *For any set of jobs J, m identical machines, and $i \leq m$ let S_i be a schedule of J on i identical machines optimal with respect to $\sum C_j$. Then $\sum C_j(S_1) - \sum C_j(S_2) \geq \sum C_j(S_2) - \sum C_j(S_3) \geq \ldots \geq \sum C_j(S_{m-1}) - \sum C_j(S_m)$.*

Proof. Assume for simplicity that $|J|$ is divisible by $i(i+1)(i+2)$. If this is not the case, then we add dummy jobs with $p_j = 0$; obviously, this does not increase $\sum C_j$ since we can always move them to the beginning of their machines.

Fix the ordering of jobs with respect to nonincreasing processing times. Now we may associate with each job its multiplier corresponding to its position on its machine. If a job j_h has a multiplier l, then it contributes lp_h to $\sum C_j$, and it is scheduled as the l-th last job on a machine.

Now think of the multipliers in the terms of blocks of size $i+1$. For S_i the multipliers with respect to job order are:

$$\underbrace{1,\ldots,1,1,2}_{\text{The first block}};\underbrace{2,\ldots,2,3,3}_{\text{The second block}};\ldots;\underbrace{i,\ldots,i+1,i+1,i+1}_{\text{The (i)-th block}};\ldots$$

For S_{i+1} the multipliers are:

$$\underbrace{1,\ldots,1,1,1}_{\text{The first block}};\underbrace{2,\ldots,2,2,2}_{\text{The second block}};\ldots;\underbrace{i,\ldots,i,i,i}_{\text{The (i)-th block}};\ldots$$

For S_{i+2} the multipliers are:

$$\underbrace{1,1,1,\ldots,1}_{\text{The first block}};\underbrace{1,2,2,\ldots,2}_{\text{The second block}};\ldots;\underbrace{i-1,\ldots,i-1,i,i}_{\text{The (i)-th block}};\ldots$$

Also, let the sum of multipliers of the k-th block in S_i be s_k^i.

By some algebraic manipulations we prove that

$$\begin{aligned} s_k^i &= (i+1)k + k + \lfloor (k-1)/i \rfloor, \\ s_k^{i+1} &= (i+1)k, \\ s_k^{i+2} &= (i+1)k - k + \lfloor k/(i+2) \rfloor. \end{aligned}$$

It follows directly that $s_{k-1}^i - s_{k-1}^{i+1} \geq s_k^{i+1} - s_k^{i+2}$ for $k \geq 2$.

The smallest processing time in the k-th block is at least $p_{(i+1)k}$, and by the ordering of jobs $p_{(i+1)k} \geq p_{(i+1)k+1}$, therefore the contribution of the k-th block to $\sum C_j(S_i) - \sum C_j(S_{i+1})$ is at least $p_{(i+1)k+1}(s_k^i - s_k^{i+1})$. Similarly, the largest processing time in the $(k+1)$-th block is at most $p_{(i+1)k+1}$ so the contribution of the $(k+1)$-th block to $\sum C_j(S_{i+1}) - \sum C_j(S_{i+2})$ is at most $p_{(i+1)k+1}(s_k^{i+1} - s_k^{i+2})$. Thus the contribution of the $(k+1)$-th block to $\sum C_j(S_{i+1}) - \sum C_j(S_{i+2})$ is at most the contribution of the k-th block to $\sum C_j(S_i) - \sum C_j(S_{i+1})$, for all $k \geq 1$. Also, the first block does not contribute to $\sum C_j(S_{i+1}) - \sum C_j(S_{i+2})$, which proves the lemma. □

Corollary 1. *For a given instance of the problem $P|G = \textit{complete multipartite}|\sum C_j$ a schedule constructed by the greedy assignment has optimal $\sum C_j$. The method can be implemented in $\mathrm{O}(mn + n\log n)$ time.*

Proof. Let S_{alg} and S_{opt} be the greedy and optimal schedules, respectively. If the numbers of machines assigned to each of the parts are equal in S_{alg} and S_{opt}, then the theorem obviously holds.

Assume that there is a part J_i that has more machines assigned in S_{opt} than in S_{alg}. It means that there is also a part J_j that has fewer machines assigned in S_{opt} than in S_{alg}.

Let us construct a new schedule S_{opt} by assigning one more machine to J_i and one less to J_j. By Lemma 1 and by the fact that greedy method added a machine to J_i instead of J_j, we decreased $\sum C_j$ on part J_i no less than we increased it on part J_j. Hence, the claim follows.

The complexity follows from the fact that we may calculate the initial and prospective $\sum C_j$ for each part and store the difference (possible saving) in a heap. A greedy assignment is equivalent to taking the highest saving, and recalculating the possible saving for the corresponding part. □

Uniform Machines

In this section we prove a series of results for various problems on uniform machines. In particular, we start by showing that both $Q|G = \textit{complete multipartite}, p_j = 1|C_{\max}$ and $Q|G = \textit{complete multipartite}, p_j = 1|\sum C_j$ are Strongly NP-hard. Moreover, we found 2-approximation and 4-approximation algorithms for the first and the second problem, respectively.

On the other hand, if we make a number of parts a part of the problem, not of the input, we find that both $Q|G = \textit{complete k-partite}, p_j = 1|C_{\max}$ and $Q|G = \textit{complete k-partite}, p_j = 1|C_{\max}$ can be solved in polynomial time. Finally, we extend our analysis beyond the unit length tasks and provide a 4-approximation algorithm for $Q|G = \textit{complete k-partite}|C_{\max}$ problem, based on linear programming.

Theorem 1. *$Q|G = \textit{complete multipartite}, p_j = 1|\sum C_j$ is* Strongly NP-hard.

Proof. We proceed by reducing Strongly NP-complete 3-Partition (Garey and Johnson 1979) to our problem.

Recall that an instance of 3-Partition is (A, b, s'), where A is a set of $3m$ elements, b is a bound value, and s' is a size function such that for each $a \in A$, $\frac{b}{4} < s'(a) < \frac{b}{2}$ and $\sum_{a\in A} s'(a) = mb$. The question is whether A can be partitioned into disjoint sets $A_1, \ldots, A_m$, such that $\forall_{1\le i\le m} \sum_{a\in A_i} s(a) = b$.

For any (A, b, s') we let $G = (J_1 \cup \ldots \cup J_m, E) = \textit{complete m-partite}$, where $|J_i| = b$ for all $i = 1, 2, \ldots, m$. Moreover, let $M = \{m_1, \ldots, m_{3m}\}$ with speeds $s(m_i) = s'(a_i)$. Finally, let the limit value be $\sum C_j = \frac{m(b+1)}{2}$.

Suppose now that an instance (A, b, s') admits a 3-partition and let the sets be $A_1, \ldots, A_m$. Then if $a_i \in A_j$, we assign exactly $s'(a_i)$ jobs from J_j to the machine m_i. Since for every i it holds that $\sum_{a\in A_i} s'(a) = b$, we know that all jobs are assigned. Moreover, we never violate the incompatibility graph conditions, as we assign to any machine only jobs from a single part.

By assigning $s'(a_i)$ jobs to a machine m_i we ensure that

$$\sum C_j = \sum_{i=1}^{|M|} \frac{\binom{s'(a_i)+1}{2}}{s(m_i)} = \sum_{i=1}^{3m} \frac{s'(a_i)+1}{2} = \frac{m(b+1)}{2}.$$

Conversely, suppose that we find a schedule S with $\sum C_j \le \frac{m(b+1)}{2}$. Now, let l_i be the number of jobs assigned to m_i in S. Let us consider the following quantity:

$$X := \sum_{i=1}^{|M|} \binom{l_i+1}{2} \frac{1}{s(m_i)} - \frac{m(b+1)}{2}$$
$$= \sum_{i=1}^{|M|} \frac{l_i + s(m_i) + 1}{2s(m_i)} (l_i - s(m_i)).$$

X is the difference between $\sum C_j(S)$ and $\sum C_j$ of a schedule, where each machine m_i is assigned $s(m_i)$ jobs. Now, we note that $\sum_{i=1}^{|M|}(l_i - s(m_i)) = 0$ as every job is assigned somewhere. Moreover,

$$l_i + s(m_i) + 1 > 2s(m_i) \quad \text{if } l_i \ge s(m_i),$$
$$l_i + s(m_i) + 1 \le 2s(m_i) \quad \text{if } l_i < s(m_i).$$

By combining the last two facts, we note that every element $l_i - s(m_i) \ge 0$ in X gets multiplied by some number greater than 1, and every $l_i - s(m_i) < 0$ gets multiplied by some number not greater than 1. Therefore $\sum_{i=1}^{|M|}(l_i - s(m_i)) = 0$ implies $X \ge 0$. Moreover, if there exists any element, such that $l_i - s(m_i) > 0$, then $X > 0$. However, a schedule with $\sum C_j \le \frac{m(b+1)}{2}$ satisfies $X \le 0$, therefore it holds that $X = 0$ and $l_i = s(m_i)$ for all machines.

Each machine has jobs from exactly one part assigned. Let M_j be the set of machines on which the jobs from J_j are executed. By the previous argument, m_i has exactly $s(m_i)$ jobs assigned in S. By this and by the bounds on $a \in A$, we have $|M_j| = 3$. By the correctness of the schedule all jobs are run on some machines, so the division into M_1, M_2, …, M_m corresponds to a partition. □

Theorem 2. *$Q|G = \textit{complete multipartite}, p_j = 1|C_{\max}$ is* Strongly NP-hard.

Proof. The proof is almost identical to that of Theorem 1: we transform an input 3-Partition instance to our problem in the same way. Then we use 1 as the limit on $C_{\max}$, which similarly ensures that any machine with speed $s(m)$ gets exactly $s(m)$ unit jobs in a schedule of such a makespan. Such a schedule again corresponds directly to a solution for the instance of 3-Partition. □

Now we turn to the search for good worst-case approximations that can be obtained in polynomial time. It turns out that by using a similar approach we can construct a 2-approximate algorithm for $Q|G = \textit{complete multipartite}, p_j = 1|C_{\max}$ and a 4-approximate algorithm for $Q|G = \textit{complete multipartite}, p_j = 1|\sum C_j$.

We begin by defining an auxiliary Number Covering problem as follows: The instance is given as $(A, s_1, \ldots, s_k)$, where A is a multiset of natural numbers, and $s_1, \ldots, s_k$ are natural numbers. The numbers and the set A are such, that there exists a division of A into k multisets $A_1, \ldots, A_k$ for which $\forall_{i\in 1,\ldots,k}\ \ s_i \le \sum_{a\in A_i} a$. Given an instance $(A, s_1, \ldots, s_k)$ we want to find $F : A \to \{1, \ldots, k\}$ with $f_i = \sum_{a:F(a)=i} a$ such that $\min_{i=1}^{k} \frac{\min\{s_i, f_i\}}{s_i}$ is maximal. Note that the optimal solution has the value 1, due to the condition on A and $s_1, \ldots, s_k$.

However, finding an optimal function is NP-hard as it can be used to solve 3-Partition problem so we focus on constructing an approximation algorithm.

Lemma 2. *Let the* greedy algorithm *for* Number Covering *be as follows*: *order s_i in such a way that $s_1 \geq \ldots \geq s_k$. Then, consider the numbers in $a \in A$ ordered non-increasingly and starting from $i = 1$ assign $F(a) = i$ (i.e., cover s_i) until $f_i \geq \frac{1}{2}s_i$ holds. When it holds proceed to $i + 1$, until k. The greedy algorithm is $\frac{1}{2}$–approximation algorithm for* Number Covering.

Proof. First, let us prove a particular case: $s_i = \sum_{a \in A_i} a$. We establish the invariant that after covering of each s_i greedily with $f_i \geq \frac{1}{2}s_i$ the sum of the remaining numbers from A remains no smaller than $\sum_{j=i+1}^{k} s_i$. If this condition holds, it is always possible to assign $F(a) = i + 1$ to the largest remaining numbers from A until we get $f_{i+1} \geq \frac{1}{2}s_{i+1}$ – so by induction this algorithm guarantees $f_i \geq \frac{1}{2}s_i$ for all i.

At the beginning of the algorithm, i.e., for $i = 0$ the invariant obviously holds. Now suppose that the invariant holds for all values $1, \ldots, i-1$ and let us use the remaining numbers from A in a non-increasing fashion to cover s_i.

If we get $s_i \geq f_i \geq \frac{1}{2}s_i$ the invariant is preserved since we decreased the sum of the remaining numbers in A by f_i and we decreased the sum of numbers to be covered by s_i.

Assume that we get $f_i > s_i$ and let $a' = \min\{a : F(a) = i\}$. Observe that s_i had to be covered by exactly one element – for otherwise it would contradict the order of the elements, as $0 < f_i - a' < \frac{1}{2}s_i$ so $f_i - a' < a'$. Therefore all remaining elements in A after covering s_i certainly include all the numbers from $A_{i+1} \cup \ldots \cup A_k$ as we did not use any element from A less than or equal to s_i, and the sets can consist of only these elements. Hence, the invariant again holds.

Finally, in the general case $s_i \leq s'_i = \sum_{a \in A_i} a$ it is sufficient to note that we may treat any *lowering* of s'_i to s_i as an ordering which does not change their relative ordering. For example, if for $s' = (10, 8, 6)$ we have $s = (5, 8, 6)$, then we process it as $(8, 6, 5)$, which is also a lowering of s'. Therefore, if the greedy strategy covers $\frac{1}{2}$–approximately s', then it has to cover s within the same approximation ratio. □

Theorem 3. *There exists a 2-approximate algorithm for $Q|G =$ complete multipartite$, p_j = 1|C_{\max}$.*

Proof. For a given bound on $C_{\max}$ equal to T, we know that we can schedule on m_i at most $\lfloor s(m_i) \cdot T \rfloor$ jobs. Therefore we construct $A = \{\lfloor s(m_i) \cdot T \rfloor : i = 1, \ldots, m\}$ and $s_i = |J_i|$ for $i = 1, \ldots k$. Note that at this point we do not know whether it is a proper instance of Number Covering or not.

Now we run greedy algorithm on $(A, s_1, \ldots, s_k)$ and we may get two results:

1. success – the greedy algorithm returned a $\frac{1}{2}$-covering, i.e., F such that $f_i \geq \frac{1}{2}s_i$ for all $i = 1, \ldots k$,
2. failure – the greedy algorithm returned a noncovering, i.e., F such that $f_i < \frac{1}{2}s_i$ for some $i = 1, \ldots k$.

By Lemma 2 the second case cannot occur if $(A, s_1, \ldots, s_k)$ is an instance of Number Covering. Let OPT be $C_{\max}$ of an optimum schedule. If $T \geq OPT$ it is always true that $(A, s_1, \ldots, s_k)$ is an instance of Number Covering: all graph parts (with sizes s_i) can be covered completely by some disjoint subsets of machines (with capacities a_j) – so we are sure that we get a success. Hence, using binary search we can find the smallest T for which covering succeeds, and T is guaranteed to be at most OPT.

Now, let us take $\frac{1}{2}$-covering F for this T. We translate it to a schedule as follows: for $j = 1, \ldots, m$ if $a_j = \lfloor s(m_j) \cdot T \rfloor$ has $F(a_j) = i$, we schedule up to $2a_j$ jobs from J_i on m_j. In total, there are up to $2f_i \geq s_i = |J_i|$ jobs scheduled for every J_i, which ensures that all jobs are scheduled somewhere. Moreover, each machine gets jobs only from a single part. Finally, it is clear that the maximum completion time of this schedule is at most $2\,T \leq 2\,OPT$. □

Interestingly, the above algorithm can be modified in a non-trivial way to give 4-approximation for $\sum C_j$. If we knew the number of jobs assigned to each machine in some optimal schedule, then it would be again reduced to $\frac{1}{2}$-approximation of Number Covering. However, it turns out that this implies a special way of assigning machines to parts of the graph – and we can use it to solve the general problem.

Theorem 4. *There exists a 4-approximate algorithm for $Q|G =$ complete multipartite$, p_j = 1|\sum C_j$.*

Proof. Suppose that we knew in advance the number of jobs l_i assigned to a machine m_i in some optimal schedule. Then we could construct $A = \{l_i : i = 1, \ldots, m\}$ and $s_i = |J_i|$ as an instance of Number Covering. By Lemma 2 there is $\frac{1}{2}$-covering F. We could translate it to an approximate schedule as follows: if $a_j = l_j$ has $F(a_j) = i$, then assign up to $2a_j$ jobs from J_i to m_j. In total, there are up to $2f_i \geq s_i = |J_i|$ jobs scheduled for every J_i, so every job is scheduled somewhere. Now observe that m_i in an optimal schedule contributes exactly $\binom{l_i+1}{2}\frac{1}{s(m_i)}$ to $\sum C_j$, but in approximate schedule it contributes $\binom{2l_i+1}{2}\frac{1}{s(m_i)} \leq 4\sum C_j(m_i)$. So this would be a 4-approximate schedule.

Assume that the parts and machines are sorted in order of their nonincreasing sizes and speeds, respectively. Notice, that without loss of generality the numbers of the jobs the ordered machines process are nonincreasing in an optimal schedule. Hence, the previous argument implies that there exists a 4-approximate schedule in which machines are assigned to parts, in exactly this order. Precisely, J_1 gets the fastest l_1 machines, J_2 gets the next fastest l_2 machines, etc. Let us call by *ordered assignment* any such assignment from machines to parts as long as $\sum_{i=1}^{k} l_i \leq m$. Notice that, in a sense, an assignment of machines to parts determines a corresponding schedule.

This allows us to construct a 4-approximation algorithm even if we do not know l_i. We proceed by using dynamic programming over all ordered assignments. Precisely, let the states of this program be defined by $(j, i, cost)$. It corresponds to an ordered assignment of the first i machines to the first j parts optimally with respect to $\sum C_j$. Notice that $(1, 1, cost), \ldots, (1, m - (k-1), cost)$ are well defined. For $k' \geq 2$, $m - (k - k') \geq m' \geq k'$ and

$m' - 1 \geq m'' \geq k' - 1$, construct $(k', m', cost)$ as the best with respect to $\sum C_j$ of the ordered assignments corresponding to $(k' - 1, m'', cost'')$ and the assignment of the rest of $m' - m''$ first machines to $J_{k'}$. Every such an assignment is feasible. Moreover, for any $1 \leq k' \leq k$ and $k' \leq m' \leq m - (k' - k)$ holds that there is no partial ordered assignment with smaller $\sum C_j$ than $cost$ following from $(k', m', cost)$. Consider a counter-example with the minimum number of the parts and the minimum number of the machines. In this case, let there be an ordered assignment O_{opt} of minimum $\sum C_j$ defined by the numbers of the machines assigned to $J_1, \ldots, J_k$, and let these numbers be $n_1, \ldots, n_k$, respectively. Consider an ordered assignment O_{alg} determined by $(k-1, n_1 + \ldots + n_{k-1}, cost)$ and $\sum C_j$ determined by the assignment of last n_k machines to J_k. Notice that the contributions of last n_k machines to $\sum C_j$ are equal in O_{opt} and O_{alg}, hence there is a minimum counter-example on $k-1$ parts and $n_1 + \ldots + n_{k-1}$ machines.

At each step of our dynamic program there are at most mn states since for every (i, j) we store only the smallest c. There are up to m possible new assignments generated from each state, each requiring $\mathrm{O}(n \log m)$ time to generate. Therefore each step requires $\mathrm{O}(m^2 n^2 \log m)$ operations and the total running time of the algorithm is $\mathrm{O}(km^2 n^2 \log m)$. □

When the number of parts is fixed, we show that there are polynomial algorithms for solving the respective problems for both $C_{\max}$ and $\sum C_j$ criteria.

Theorem 5. *There exists a* $\mathrm{O}(mn^{k+1} \log(mn))$ *algorithm for* $Q|G =$ *complete k-partite*, $p_j = 1|C_{\max}$.

Proof. We adopt the framework of Hochbaum and Shmoys (1988), i.e., we guess $C_{\max}$ of a schedule and check whether this is a feasible value. There are only up to $\mathrm{O}(mn)$ possible values of $C_{\max}$ to consider, as it has to be determined by the number of jobs loaded on a single machine, and we can use binary search to find the optimal value.

Now assume that we check a single candidate value for $C_{\max}$. Fix any ordering of the machines. We store the information if there exists a feasible assignment of the first $0 \leq l \leq m$ machines such that there are a_i unassigned jobs for part J_i. In each step there is a set of tuples $(a_1, \ldots, a_k)$ corresponding to the remaining jobs of $\mathrm{O}(n^k)$ size. We start the algorithm for $l = 0$ with $(|J_1|, \ldots, |J_k|)$ – if no machines are used, then all jobs are unassigned.

For $l \geq 1$ we take all tuples $(a_1, \ldots, a_k)$ from the previous iteration. For each tuple we try all possible assignments of m_l to the parts, then we try all feasible assignments of the remaining jobs to this machine. This produces updated tuples, determining some feasible assignment of the first l machines. For each $(a_1, \ldots, a_k)$ we construct at most kn updated tuples, each in time $\mathrm{O}(1)$, so for constant k the work is bounded by $\mathrm{O}(n^{k+1})$. Note that we do need to store only one copy of each distinct $(a_1, \ldots, a_k)$.

After considering $l = m$ machines it is sufficient to check if the tuple $(0, 0, \ldots, 0)$ is feasible. Clearly the total running time for a single guess of $C_{\max}$ is $\mathrm{O}(mn^{k+1})$. □

Theorem 6. *There exists a* $\mathrm{O}(mn^{k+1})$ *algorithm for* $Q|G =$ *complete k-partite*, $p_j = 1|\sum C_j$.

Proof. Let us process machines in any fixed ordering. Let the state of the partial assignment be identified by a tuple $(a_1, \ldots, a_k, c)$, where a_i denotes the number of vertices remaining to be covered and c denotes $\sum C_j$ of the jobs that have been scheduled so far.

Assume that two partial assignments P_1 and P_2 on m' first machines are described by the same state $(a_1, \ldots, a_k)$; and two values c_1 and c_2, respectively. If $c_2 \geq c_1$, then any extension of P_2 on $m'' > m'$ first machines cannot be better than the exactly the same extension of P_1 on m'' machines. Therefore for any $(a_1, \ldots, a_k)$ it is sufficient to store only the tuple $(a_1, \ldots, a_k, c)$ with the smallest c.

We may proceed with a dynamic program similar to the one used for $C_{\max}$. That is, we start with a single state $(|J_1|, \ldots, |J_k|, 0)$. In the l-th step ($l = 1, \ldots, m$) we take states $(a_1, \ldots, a_k, c)$, corresponding to feasible assignments for the first $l-1$ machines. We try all k possible assignments of m_l to parts. If m_l is assigned to J_i, for some i, then we try all choices of the number $n' \in \{0, \ldots, a_i\}$ of remaining jobs from J_i. Such a choice together with the assignment of m_l determines an assignment of n' unassigned jobs to m_l. If the tuple constructed $(a'_1, \ldots, a'_k, c')$ has $\sum C_j$ inferior to the already produced we do not store it. Finally, after considering all machines we obtain exactly one tuple of the form $(0, \ldots, 0, c)$, which determines the optimal schedule.

At each step there are at most n^k states since for every $(a_1, \ldots, a_k)$ we store only the smallest c. There are up to n possible new assignments generated from each state. Each try requires $\mathrm{O}(k)$ time. Therefore, for any fixed k the time complexity of the algorithm is $\mathrm{O}(mn^{k+1})$. □

Now we turn our attention to the problem with a bounded number of parts of the graph, but where tasks have arbitrary processing requirements. First, we prove that given a set of machines with particular speeds we can merge them to one faster machine not increasing $\sum C_j$ of an optimal schedule. Using this observation, exhaustive search, linear programming, and rounding we are able to prove Theorem 7.

Lemma 3. *Let* J *be any set of jobs. Let* M *be a set of uniform machines. Then it holds that* $\sum C_j$ *of an optimal schedule of* J *on* M *is at least as big as the optimal* $\sum C_j$ *for* J *and a single machine with speed equal to* $\sum_{m \in M} s(m)$.

Proof. We prove the case for $M = \{m_1, m_2\}$ with speeds $s_1 \geq s_2$. For a higher number of machines the lemma follows by induction on the number of the machines.

Consider an optimal schedule on the machines. It is equivalent to two steps. First, the selection of n smallest multipliers of processing requirements. Second, an assignment of the jobs ordered non-increasingly with respect to processing requirements, to positions on the machines, corresponding to the multipliers. Notice that the sequence of the n smallest multipliers are w.l.o.g. (by $\frac{1}{s_1} \leq \frac{1}{s_2}$) of the following form:

$$\underbrace{\frac{1}{s_1}, \ldots, \frac{n_1}{s_1}}_{n_1}; \frac{1}{s_2}; \underbrace{\frac{n_1+1}{s_1}, \ldots, \frac{n_1+n_2}{s_1}}_{n_2}; \frac{2}{s_2}; \ldots$$

The number n_i is the maximum number of multipliers from the m_1, that were not used previously, and that are less than or equal to i-th multiplier from m_2. The multipliers form a sequence of blocks of sizes $n_1+1, n_2+1, \ldots$, respectively. For m' with $s(m') = s(m_1) + s(m_2) = s_1 + s_2$ the multipliers are of the following form:

$$\underbrace{\frac{1}{s_1+s_2}, \ldots;}_{n_1} \frac{n_1+1}{s_1+s_2}; \underbrace{\frac{n_1+2}{s_1+s_2}, \ldots;}_{n_2} \frac{n_1+n_2+2}{s_1+s_2}; \ldots$$

It is sufficient to prove that the i-th multiplier in the first sequence is at least as large as in the second one.

Let us prove that it holds for k-th block for any $k = 1, 2, \ldots$: for the first multiplier in the block it holds that $\frac{\sum_{i=1}^{k-1} n_i+1}{s_1} \geq \frac{\sum_{i=1}^{k-1} n_i+(k-1)+1}{s_1+s_2}$, by the fact that $\frac{\sum_{i=1}^{k-1} n_i+1}{s_1} \geq \frac{k-1}{s_2}$. Here we used the fact that if $a \geq b$, then $a \geq pa+(1-p)b$ for any $0 \leq p \leq 1$. For all the next $n_k - 1$ multipliers it holds by a simple inductive argument, since we add at each step $\frac{1}{s_1} > \frac{1}{s_1+s_2}$. Finally, for the last element in the block, $\frac{k}{s_2} \geq \frac{\sum_{i=1}^{k} n_i+k}{s_1+s_2}$ holds, since $\frac{k}{s_2} \geq \frac{\sum_{i=1}^{k} n_i}{s_1}$ – by the same argument as for the first element. □

Roughly speaking, the main idea of the following algorithm lies in the fact that for each of the parts, we may guess the speed of the fastest machine and the number of such machines. Then we construct a linear (possibly fractional) relaxation of the assignment of the machines to the parts and round down the numbers of the machines assigned to the nearest integer. To show that rounding down does not increase $\sum C_j$ more than 2 times, we use Lemma 3 and properties of geometric series, to bound the profit from hypothetical rounding up. This together with the rounding of speeds of the machines proves the following theorem.

Theorem 7. *There exists a 4-approximation algorithm for $Q|G = \text{complete k-partite}|\sum C_j$.*

Proof. Consider Algorithm 1. First notice that the proposed program is an LP relaxation of the scheduling problem. Precisely, $n_{pr,tp}$ means how many machines from a group tp are assigned to the part pr; $x_{jb,lr,tp}$ means what part of a job jb is assigned as the lr-th last on a machine of type tp. Notice that jobs assigned to machines of a given type form layers, i.e., jobs assigned as last contribute their processing times once, as the last by one contributes twice, etc.

About the conditions:

- Condition (1) guarantees that all the machines are assigned, fractionally at worst.
- Condition (2) provides that no machine with speed higher than maximum possible (i.e., guessed) is assigned to the part.
- Condition (3) guarantees that each of the parts can be given any number of not preassigned (not assigned by guessing) machines of a given type.
- Condition (4) guarantees that the given number of machines of guessed type is assigned to a given part as the fastest ones.

Algorithm 1 4-approximate algorithm for the problem $Q|G = \text{complete k-partite}|\sum C_j$

Require: $J = (J_1, \ldots, J_k)$, $M = \{m_1, \ldots, m_m\}$.

1: Round the speeds of the machines up to the nearest multiple of 2.
2: Let the nonempty group of the machines, ordered by the speeds be $M_1, \ldots, M_l$.
3: For each part J_{pr}, guess the speed s'_{pr} and the number n'_{pr} of fastest machines assigned to it in an optimal schedule. Discard unfeasible guesses.
4: Solve the linear program with variables:
   - $n_{pr,tp}$, where $pr \in \{1, \ldots, k\}$, $tp \in \{1, \ldots, l\}$,
   - $x_{jb,lr,tp}$, where $jb \in J_1 \cup \ldots, \cup J_k$, $lr \in \{1, \ldots, n\}$, $tp \in \{1, \ldots, l\}$.
5: Let the LP conditions be:

$$\sum_{pr} n_{pr,tp} = |M_{tp}| \qquad \forall tp \quad (1)$$

$$n_{pr,tp} = 0 \quad \forall pr \forall tp > s'_{pr} \quad (2)$$

$$n_{pr,tp} \leq |M_{tp}| - \sum_{i \in \{i | s'_i = tp\}} n'_i \quad \forall pr \forall tp < s'_{pr} \quad (3)$$

$$n_{pr,tp} = n'_{pr} \quad \forall pr, tp = s'_{pr} \quad (4)$$

$$\sum_{lr,tp} x_{jb,lr,tp} = 1 \qquad \forall jb \quad (5)$$

$$\sum_{jb \in J_{pr}} x_{jb,lr,tp} \leq n_{pr,tp} \qquad \forall pr \forall tp \forall lr \quad (6)$$

$$0 \leq x_{jb,lr,tp} \qquad \forall jb \forall lr \forall tp \quad (7)$$

$$0 \leq n_{pr,tp} \qquad \forall pr \forall tp \quad (8)$$

6: Let the LP cost function be: $\sum_{jb,lr,tp} x_{jb,lr,tp} \cdot lr \cdot p(jb) \cdot \frac{1}{s(tp)}$, where $p(jb)$ is the processing requirement of job jb and $s(tp)$ is the speed factor of machine of type tp.
7: Solve the jobs assignment for each part separately using the optimal solution of LP.

- Condition (5) ensures that any job is assigned completely, in a fractional way at worst.
- Condition (6) guarantees that for a given layer, part, and machine type there are no more jobs assigned than the machines of this type to the part.

The cost function corresponds to an observation that a job jb assigned as the l-th last on the machine of type tp contributes exactly $\frac{l \cdot p(jb)}{s(tp)}$ to $\sum C_j$.

An optimal solution to LP (x^*, n^*) corresponds to a fractional assignment of machines to the parts.

We now construct for each part J_i separately a *part fractional scheduling* problem in the following way:

- The new set of variables $y_{jb,lr,m}$ indicating a fractional assignment of $jb \in J_i$ as the lr-th last job on machine $m \in M'$.
- The cost function $\sum_{jb,lr,m} y_{jb,lr,m} \cdot \frac{lr \cdot p(jb)}{s(m)}$.
- The conditions:

- $\forall_{jb,lr,m}\ y_{jb,lr,m} \geq 0$,
- $\forall_{jb} \sum_{lr,m} y_{jb,lr,m} = 1$ – each job has to be assigned completely,
- $\forall_{lr,m} \sum_{jb} y_{jb,lr,m} \leq 1$ – each layer on each machine cannot contain more than a full job in total.

Here the set of machines M' consists of exactly $\lceil n^*_{i,tp} \rceil$ machines for each $1 \leq tp \leq l$. Hence, for each type we add at most one „virtual" machine due to rounding, except the machine with the highest speed per part, which were preassigned exactly.

Now we rearrange jobs within layers for machines of the same speed to construct some feasible solution to part fractional scheduling. Hence, let $Y = \{y_{jb,lr,m} : s(m) = tp, jb \in J_i\}$, for any fixed lr and tp. We redistribute $x_{jb,lr,tp}$ in the following way: $\forall_{lr,tp}$ we set $y_{jb,lr,m} = x_{jb,lr,tp}$ for the consecutive variables $x_{jb,lr,tp}$. If such an assignment would set some variable $y_{jb,lr,m} = y'$ such that $\sum_{jb} y_{jb,lr,m} > 1$, then we set $y_{jb,lr,m} = x_{jb,lr,tp} - (y' - 1)$, instead. And we continue with the next machine of speed tp and the unassigned fraction of $x_{jb,lr,tp}$. Notice that by condition (6) we have $\forall_{lr,tp} \sum_{jb} x_{jb,lr,tp} \leq n^*_{i,tp} \leq |Y|$. Since we only rearrange jobs preserving their layers the cost of y in part fractional scheduling is equal to the contribution of variables from J_i to the cost of x^* in LP. Hence an optimal solution can have only at most this cost.

Let us model this LP as a flow network. We construct: a set of vertices $V = J_j \cup (M' \times |J_j|)$, a set of arcs $J_j \times (M' \times |J_j|)$ with capacity 1 each, and with the cost of the flow by an arc $(jb, (m, lr))$ equal to $lr \cdot p(jb) \cdot \frac{1}{s(m)}$. Any fractional solution corresponds to a fractional flow by the network, i.e., a value of $y_{jb,m,lr}$ is exactly the flow by the arc (jb, lr, m). It is known that e.g., Successive Shortest Path Algorithm (Ahuja, Magnanti, and Orlin 1993) finds an integral minimum cost flow in such a network.

A flow in the constructed network corresponds directly to the solution to part fractional scheduling - we can treat the flow as an assignment for all the jobs in J_i. This solution is at least as good as the solution for the global relaxation of the scheduling problem.

Due to rounding of n^* there might be some non-fastest virtual machines assigned to each part, at most one per type. Using Lemma 3 combined with the fact that speeds of the machines are equal to powers of 2 we can always merge all of them and any real (preassigned) fastest machine (of speed s'_i) to a single virtual machine with a speed no greater than $2s'_i$. Hence, by scheduling all the jobs assigned to this virtual machine on the corresponding real machine of speed s'_i we increase $\sum C_j$ of these jobs at most 2 times. This together with rounding of machine speeds allows to bound the approximation ratio by 4. □

Unrelated Machines

In this section we prove that there is no good approximation algorithm possible in the case of unrelated machines.

Theorem 8. *There is no constant approximation ratio algorithm for* $R|G =$ *complete 2-partite*$|\sum C_j$ ($R|G =$ *complete 2-partite*$|C_{\max}$), *unless* P = NP.

Proof. Assume that there is a d-approximation algorithm for $R|G =$ *complete 2-partite*$|\sum C_j$ ($R|G =$ *complete 2-partite*$|C_{\max}$). Consider an instance of 3-SAT with the set of variables V and the set of clauses C, where for each $v \in V$ there are at most 5 clauses containing v. This version is still NP-complete (Garey and Johnson 1979). We construct the corresponding scheduling instance as follows. Let $M = \{v^T, v^F : v \in V\}$. Also, let $G =$ *complete 2-partite* with parts $J_1 = \{j_{v,1} : v \in V\} \cup \{j_c : c \in C\}$ and $J_2 = \{j_{v,2} : v \in V\}$. Hence $n = 2|V| + |C| \leq 7|V|$, by $|C| \leq 5|V|$.

Let $p_j = 1$ for all jobs. Let $s_1 \geq 1$ be a value determined by an instance of 3-SAT, but polynomially bounded by the size of the instance. Let now $s(j_{v,1}, v^T) = s(j_{v,2}, v^T) = s(j_{v,1}, v^F) = s(j_{v,2}, v^F) = s_1$, for any $v \in V$. Let for any $c \in C$ $s(j_c, v^T) = s_1$ if v appears in c, and $s(j_c, v^F) = s_1$ if $\neg v$ appears in c. Set all the others $s(j, m)$ to 1.

Consider an instance of the scheduling problem, corresponding to an instance of 3-SAT with answer YES. Then we can schedule J on the machines according to fulfilling valuation, in the following way: If v has value T then we assign v^T to J_1 and v^F to J_2, otherwise we assign v^F to J_1 and v^T to J_2. Hence any job in J_2 can be processed with speed s_1 similarly for any job in J_1. Hence, $\sum C_j \leq \binom{n+1}{2} \frac{1}{s_1} \leq \frac{7|V|(7|V|+1)}{s_1}$ ($C_{\max} \leq \frac{n}{s_1} \leq \frac{7|V|}{s_1}$), for an optimal schedule. Now it is sufficient to set $s_1 = 7d|V|(7|V| + 1) + 1$ ($s_1 = 7d|V| + 1$) to prove the theorem.

On the other hand, assume that the answer for an instance of 3-SAT is NO. Assume that there exists a schedule with $\sum C_j < 1$ ($C_{\max} < 1$). Assume that there is a part such that both v^T and v^F have no jobs from it assigned in the schedule. Then $\sum C_j \geq 1$ ($C_{\max} \geq 1$), a contradiction. Thus assume that $j_c \in J_1$ is a job assigned to some machine m with $s(j_c, m) = 1$. Clearly, also in this case we have a contradiction. Hence, each $j_c \in J_1$ is assigned to a machine corresponding to a valuation of the variable fulfilling c, so there exists a fulfilling valuation, a contradiction. Hence, for such an instance for any schedule $\sum C_j \geq 1$ ($C_{\max} \geq 1$).

Clearly, by using a d-approximation algorithm on an instance of the scheduling problem corresponding to a YES instance of 3-SAT we would be able to obtain a schedule with $\sum C_j < 1$ ($C_{\max} < 1$). For an instance corresponding to NO, for any schedule $\sum C_j \geq 1$ ($C_{\max} \geq 1$). Hence, we would be able to distinguish between them. □

Open Problems

The complexity status of $Q|G =$ *complete k-partite*$|\sum C_j$ and the best polynomial-time approximability of both $Q|G =$ *complete multipartite*$|\sum C_j$ and $Q|G =$ *complete multipartite*$|C_{\max}$ remain open problems. These investigations seem to be natural extensions of our research.

Recall that the status of $R|G =$ *complete 2-partite*$|\sum C_j$ and its $C_{\max}$ counterpart was settled. However, it is worth considering if there are some interesting subproblems for unrelated machines that admit exact or approximate polynomial time algorithms, assuming P ≠ NP.

Acknowledgments

This work was supported by Polish National Science Centre 2018/31/B/ST6/01294 grant and Gdańsk University of Technology, grant no. POWR.03.02.00-IP.08-00-DOK/16.

References

Ahuja, R.; Magnanti, T.; and Orlin, J. 1993. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall.

Baker, B.; and Coffman Jr, E. 1996. Mutual Exclusion Scheduling. *Theoretical Computer Science* 162(2): 225–243.

Bodlaender, H.; and Jansen, K. 1995. Restrictions of Graph Partition Problems. Part I. *Theoretical Computer Science* 148(1): 93–109.

Bodlaender, H.; Jansen, K.; and Woeginger, G. 1994. Scheduling with Incompatible Jobs. *Discrete Applied Mathematics* 55(3): 219–232.

Brucker, P. 1999. *Scheduling Algorithms*. Springer.

Bruno, J.; Coffman Jr, E.; and Sethi, R. 1974. Scheduling Independent Tasks to Reduce Mean Finishing Time. *Communications of the ACM* 17(7): 382–387.

Das, S.; and Wiese, A. 2017. On Minimizing the Makespan When Some Jobs Cannot Be Assigned on the Same Machine. In *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *LIPIcs*, 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Dessouky, M.; Lageweg, B.; Lenstra, J. K.; and van de Velde, S. 1990. Scheduling Identical Jobs on Uniform Parallel Machines. *Statistica Neerlandica* 44(3): 115–123.

Epstein, L.; and Sgall, J. 2004. Approximation Schemes for Scheduling on Uniformly Related and Identical Parallel Machines. *Algorithmica* 39(1): 43–57.

Garey, M.; and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Grage, K.; Jansen, K.; and Klein, K.-M. 2019. An EPTAS for Machine Scheduling with Bag-Constraints. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, 135–144. ACM.

Hochbaum, D.; and Shmoys, D. 1988. A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach. *SIAM Journal of Computing* 17(3): 539–551.

Horowitz, E.; and Sahni, S. 1976. Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *Journal of the ACM* 23(2): 317–327.

Jansen, K.; Lassota, A.; and Maack, M. 2020. Approximation Algorithms for Scheduling with Class Constraints. In Scheideler, C.; and Spear, M., eds., *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, 349–357. ACM.

Jansen, K.; and Maack, M. 2019. An EPTAS for Scheduling on Unrelated Machines of Few Different Types. *Algorithmica* 81(10): 4134–4164.

Kones, I.; and Levin, A. 2019. A Unified Framework for Designing EPTAS for Load Balancing on Parallel Machines. *Algorithmica* 81(7): 3025–3046.

Lawler, E.; Lenstra, J. K.; and Kan, A. R. 1982. Recent Developments in Deterministic Sequencing and Scheduling: A Survey. In Dempster, M.; Lenstra, J. K.; and Kan, A. R., eds., *Deterministic and Stochastic Scheduling*, volume 84 of *NATO Advanced Study Institutes Series* (*Series C – Mathematical and Physical Sciences*), 35–73. Springer.

Lenstra, J. K.; Shmoys, D. B.; and Tardos, É. 1990. Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Mathematical Programming* 46(1-3): 259–271.

Lonc, Z. 1991. On Complexity of Some Chain and Antichain Partition Problems. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 570 of *Lecture Notes in Computer Science*, 97–104. Springer.

Mallek, A.; Bendraouche, M.; and Boudhar, M. 2019. Scheduling Identical Jobs on Uniform Machines with a Conflict Graph. *Computers & Operations Research* 111: 357–366.

Page, D.; and Solis-Oba, R. 2020. Makespan Minimization on Unrelated Parallel Machines with A Few Bags. *Theoretical Computer Science* 821: 34–44.

Shchepin, E.; and Vakhania, N. 2005. An Optimal Rounding Gives a Better Approximation for Scheduling Unrelated Machines. *Operations Research Letters* 33(2): 127–133.

Smith, W. E. 1956. Various Optimizers for Single-Stage Production. *Naval Research Logistics Quarterly* 3(1-2): 59–66.