Total Completion Time Minimization for Scheduling with Incompatibility Cliques

Klaus Jansen,[1] **Alexandra Lassota,** [1] **Marten Maack,** [1] **Tytus Pikies** [2]
[1] Department of Computer Science, Faculty Of Engineering, Kiel University, 24098 Kiel, Germany
[2] Department of Algorithms and System Modeling, Gdańsk University of Technology, 80-233 Gdańsk, Poland
kj@informatik.uni-kiel.de, ala@informatik.uni-kiel.de, mmaa@informatik.uni-kiel.de, tytpikie@pg.edu.pl

Abstract

This paper considers parallel machine scheduling with incompatibilities between jobs. The jobs form a graph equivalent to a collection of disjoint cliques. No two jobs in a clique are allowed to be assigned to the same machine.

Scheduling with incompatibilities between jobs represents a well-established line of research in scheduling theory and the case of disjoint cliques has received increasing attention in recent years. While the research up to this point has been focused on the makespan objective, we broaden the scope and study the classical total completion time criterion. In the setting without incompatibilities, this objective is well-known to admit polynomial time algorithms even for unrelated machines via matching techniques. We show that the introduction of incompatibility cliques results in a richer, more interesting picture. We prove that scheduling on identical machines remains solvable in polynomial time, while scheduling on unrelated machines becomes APX-hard. Next, we study the problem under the paradigm of fixed-parameter tractable algorithms (FPT). In particular, we consider a problem variant with assignment restrictions for the cliques rather than the jobs. We prove that, despite still being APX-hard, it can be solved in FPT time with respect to the number of cliques. Moreover, we show that the problem on unrelated machines can be solved in FPT time for reasonable parameters, in particular, the parameter combination: maximum processing time, number of job kinds, and number of machines or maximum processing time, number of job kinds, and number of cliques. The latter results are extensions of known results for the case without incompatibilities, and can even be further extended to the case of total weighted completion time. All of the FPT results make use of n-fold Integer Programs that recently received great attention by proving their usefulness for scheduling problems.

Introduction

Consider a task system under difficult conditions like high electromagnetic radiation, or with an unstable power supply. Due to the environmental conditions, users prepare tasks in groups and want the jobs in a given group to be scheduled on different processors. That assures that even if a few processors fail, another processor will be able to execute at least part of the jobs. Due to the instability, our system even might stop working completely and in this case all jobs that are done only partially have to be scheduled again. As observed (Bruno, Jr., and Sethi 1974), the sum of completion times criterion tends to reduce the mean number of unfinished jobs at each moment in the schedule. For this reason, we would like to minimize the sum of completion times of the jobs respecting the additional reliability requirement given by the groups. In the following, we discuss the problems motivated by this scenario more formally.

Problem. In the classical problem of scheduling on parallel machines, a set J of n jobs, a set M of m machines, and a processing time function p are given. The processing times are of the form $p : J \to \mathbb{N}$ if the machines are identical or of the form $p : J \times M \to \mathbb{N} \cup \{\infty\}$ if the machines are unrelated. That is, the processing time of a job does or does not, respectively, depend on the machine to which the job is assigned to. For brevity, we usually write p_j or p_j^i instead of $p(j)$ or $p(j, i)$, for each job j and machine i. The goal is to find a schedule of the jobs on the machines, which minimizes a given objective function. A schedule in this setting is an assignment from jobs to machines and starting times. However, for brevity, by the fact that for any machine, we can order the jobs assigned to it optimally according to Smith's rule (Smith 1956), we do not specify the starting times explicitly. The completion time C_j of j is given by the sum of its starting and processing times. Probably the most studied objective function is the minimization of the makespan $C_{\max} = \max_j C_j$, directly followed by the minimization of the total completion time objective $\sum C_j$ or the sum of weighted completion times $\sum w_j C_j$. In this paper, we use the three-field notation prevalent in scheduling theory. For instance, $C_{\max}$ minimization on identical machines is abbreviated as $P||C_{\max}$ and minimization of ΣC_j on unrelated machines as $R|| \sum C_j$. For a general overview of scheduling notation we refer the reader to (Brucker 2004).

All of the scheduling problems discussed so far are fundamental and often studied with respect to additional constraints. One line of research considers incompatibilities between jobs in the sense that some jobs may not be processed by the same machine. More formally, an incompatibility graph $G = (J, E)$ is part of the input, and an edge $\{j, j'\} \in E$ signifies that in a feasible schedule j and j' cannot be assigned to the same machine. In this paper, we

problem	complexity	proposed running time
$P\|\text{cliques}\|\sum C_j$	Polynomial	$O(nm^2 + n^{5/2}m)$
$R\|\text{cliques}, M(j), (p^i_k)_{k\in[b],i\in M}\|\sum C_j$	Polynomial	$O(n^4m + n^3m^2 \log mn)$
$P\|\text{cliques}, M(k), p_i \in \{p_1 < p_2 < 2p_1\}\|\sum C_j$	APX-hard	-
$R\|\text{cliques}, p^j_i \in \{p_1 < p_2 < p_3\}\|\sum C_j$	APX-hard	-
$R\|2\text{ cliques}, p^i_j \in \{p_1 < p_2 < p_3\}\|\sum C_j$	NP-hard	-
$P\|\text{cliques}, M(k)\|\sum C_j$	FPT w.r.t. b	$2^{O(b^4\cdot\log(b))}m\log(m)\log(n)\log(mp_{\max})$
$R\|\text{cliques}\|\sum w_jC_j$	FPT w.r.t. m, $p_{\max}$, ϑ	$(p_{\max}\vartheta m)^{O(\vartheta^3m^3)}O(b\log^3(b)\log(w_{\max}))$
$R\|\text{cliques}\|\sum w_jC_j$	FPT w.r.t. b, $p_{\max}$, ϑ	$(p_{\max}\vartheta b)^{O(\vartheta^3b^3)}O(m\log^2(m)\log(w_{\max}))$

Table 1: An overview of the results presented in this paper.

study variants of $R||\sum(w_j)C_j$ in which the incompatibility graph is a collection of cliques corresponding to the groups of jobs mentioned above. In the three-field notation, we denote the class to which the incompatibility graph belongs in the middle, e.g. $R|\text{cliques}|\sum(w_j)C_j$. Some subvariants of $R|\text{cliques}|\sum(w_j)C_j$ are interesting on their own, hence we add additional information in the middle field, to distinguish them. For example, by $P|\text{cliques}, M(j)|C_{\max}$ we understand a subproblem of $R|\text{cliques}|C_{\max}$ in which the processing times are given by a function $p : J \to \mathbb{N}$, but each job j may only be processed on a given set $M(j)$ of machines eligible for j. Similarly, by $P|\text{cliques}, M(k)|C_{\max}$, we denote a variant in which jobs in a clique $k \in [b]$ have the same set of eligible machines $M(k)$. Or, we denote by $R|\text{cliques}, M(j), (p^i_k)_{k\in[b],i\in M}|\sum C_j$ a setting derived from the motivational example. In this setting, jobs belonging to the same clique have the same processing times (denoted by $(p^i_k)_{k\in[b],i\in M}$), hence they can be seen as copies of the same job. Moreover, an assignment of a job can be restricted to some machines, which is denoted by $M(j)$.

Related Work. First, note that both $P||\sum w_jC_j$ and $P||C_{\max}$ are well-known to be strongly NP-hard (Garey and Johnson 1979). On the other hand, $R||\sum C_j$ can be solved via matching techniques (Bruno, Jr., and Sethi 1974).

Scheduling with incompatibilities has first been considered in the 1990's by Bodlaender, Jansen and Woeginger (Bodlaender, Jansen, and Woeginger 1994), who studied $P||C_{\max}$ with incompatibilities between jobs in the sense used in this paper. Among other things the authors presented an approximation algorithm with approximation ratio depending on the quality of a coloring for the incompatibility graph. This result yields constant approximation algorithms for subproblem where incompatibility graph can be colored in polynomial time with with constant number of colors, which is less than the number of the machines. Furthermore, (Bodlaender and Jansen 1993) presents hardness results in the same setting for cographs, bipartite graphs and interval graphs. In (Dokka, Kouvela, and Spieksma 2012) approximation and inapproximability results are presented for the so called multi-level bottleneck assignment problem. This problem can be seen as a variant of $P|\text{cliques}|C_{\max}$ in which each clique has the same size and each machine has to receive exactly one job from each clique.

However, the exact setting studied in the present paper (with respect to incompatibilities) was introduced only recently by Das and Wiese (2017), who called the cliques *bags*. They obtained a PTAS for $P|\text{cliques}|C_{\max}$ and showed that there is no constant approximation algorithm for $P|\text{cliques}, M(j)|C_{\max}$, unless $\mathsf{P} = \mathsf{NP}$. Moreover, they gave an 8-approximation for $P|\text{cliques}, M(k)|C_{\max}$. This line of research was continued by two groups. In particular, in (Grage, Jansen, and Klein 2019) there was an EPTAS for $P|\text{cliques}|C_{\max}$ presented. The second group, considered in (Page and Solis-Oba 2020) a variant of $R|\text{cliques}|C_{\max}$ where the number of machine types and cliques is restricted and obtained a PTAS among many other results. Two machines have the same type if the processing time of each job is the same on both of them.

Finally, we also consider fixed-parameter tractable (FPT) algorithms for scheduling problems. A good overview on this line of research is provided in a survey (Mnich and van Bevern 2018). The most notable result in our context is probably a work due to Knop and Koutecký (2018), who used n-fold Integer Programs to prove (among other things) two FPT results for $R||\sum w_jC_j$. In particular, $R||\sum w_jC_j$ is FPT with respect to the number of machines and the number of different job kinds ϑ , and also FPT with respect to the maximum processing time, the number of different job kinds ϑ, and the number of distinct machine types κ. Jobs are of the same kind if they are indistinguishable regarding their properties. These results were generalized and greatly extended by Knop et al. (2019). In their work, they introduce a general framework for solving various configuration ILPs by modeling them as (an extended version of) the Monoid Decomposition problem. This allows to solve many problems with different kinds of objects (for example, jobs with release times and due dates), environments (for example, unrelated machines), and (linear or non-linear) objectives in FPT time with plenty different, natural parameterizations.

Results and Methodology. The first section begins with polynomial time algorithm for $P|\text{cliques}|\sum C_j$. Then we present a polynomial time algorithm for a setting derived from our motivational example.

Note that this setting is related to the case with clique dependent assignment restrictions introduced in (Das and Wiese 2017). Hence we study this case in the second sec-

Algorithm 1 IncompatibilitySolving(i, S)

```
Require: A set of cliques $V_1 \cup \ldots \cup V_b$, number $1 \leq i \leq m$, a schedule $S$ such that machines $m_1, \ldots, m_{i-1}$ have compatible jobs assigned.
Ensure: A schedule with the total completion time equal to the total completion time of $S$, where jobs on $m_1, \ldots, m_i$ are independent.
1: Let $V_L = \{v_L[1], \ldots, v_L[b]\}$.
2: Let $V_B = \{v_B[1], \ldots, v_B[b]\}$.
3: Construct $E$ by connecting $v_B[i]$ to $v_L[j]$ iff on $m_i, \ldots, m_m$ there is a job from $V_i$ scheduled as $j$-th.
4: Let $M'$ be a perfect matching in $(V_L \cup V_B, E)$.
5: for $l = 1, \ldots, b$ do
6:     Let $\{v_L[l], v_B[j]\} \in M'$.
7:     Exchange the job on position $l$-th on $m_i$ with any job from $V_j$ assigned to position $l$-th on $m_i, \ldots, m_m$
8: end for
9: return $S$
```

tion and prove it to be APX-hard already for the case with only two processing times $P|\text{cliques}, p_j \in \{p_1 < p_2\}, M(k)|\sum C_j$.

On the other hand, it can be solved in polynomial time if the number of cliques is constant even if there are arbitrarily many processing times. While the last result relies on costly guessing steps, we refine it in the last section using n-fold Integer Programs yielding an FPT time algorithm with respect to b for $P|\text{cliques}, M(k)|\sum C_j$. Furthermore, we revisit FPT results due to (Knop and Koutecký 2018) for $R||\sum w_j C_j$. Careful extension of the IPs considered in this work yields that $R|\text{cliques}|\sum w_j C_j$ is FPT with respect to m, $p_{\max}$, and ϑ or with respect to b, $p_{\max}$, and ϑ.

All of our results presented in this paper are summarized in Table 1. Moreover, the omitted proofs and more details can be found in the preprint (Jansen et al. 2020). We end our results with a short discussion on open problems.

Polynomial Time Algorithms

Let us begin with a key procedure for the algorithm for identical machines. In a nutshell, we prove that a perfect matching in the vertices of the graph constructed in line 4 of Algorithm 1 corresponds to a reassignment of the jobs in S in a way that the assignment of the jobs to $m_1, \ldots, m_{i-1}$ is not changed and that m_i is given a set of compatible jobs. Without loss of generality we assume that each clique V_i consists of exactly m jobs; if this is not the case we add dummy jobs with processing time 0. Notice also that in any schedule the jobs can be divided into layers. The layers are formed of the jobs that are scheduled as last on their machines, then second last, …, and as first (which correspond to b-th layer). We can exchange the jobs that are on a given layer without increasing the total completion time, because the job scheduled as last on a machine contributes once to the total completion time, the jobs scheduled as second last twice, etc.

Theorem 1 (Hall 1935). *A bipartite graph $(A \cup B, E)$ has a matching that saturates A if and only if $|N(S)| \geq |S|$ for all $S \subseteq A$.*

Lemma 1. *Let S be a schedule for an instance of $P|\text{cliques} = V_1 \cup \ldots \cup V_b|\Sigma C_j$, such that each of the machines $m_1, \ldots, m_{i-1}$ has compatible jobs assigned in S and b jobs are assigned to each machine.* Algorithm 1 *constructs in $O(bm + b^{5/2})$ time a schedule such that each of the machines $m_1, \ldots, m_i$ has compatible jobs assigned, each of the machines has b jobs and the total completion time of the new schedule is equal to the total completion time of S.*

Proof. We prove that it is always possible to exchange the jobs inside the layers $1, \ldots, b$ for machines $m_i, \ldots, m_m$, so that the load on m_i consists of compatible jobs. Consider the structure of the graph constructed in Algorithm 1. Take any nonempty $V'_B \subseteq V_B$. Notice that a clique corresponding to any vertex in V'_B has exactly $m - i + 1$ jobs on the machines $m_i, \ldots, m_m$. A layer i on the machines $m_i, \ldots, m_m$ has exactly $m - i + 1$ jobs in total. Hence, clearly the size of neighbors of V'_B in V_L is at least $|V'_B|$. If $V'_B = \emptyset$, it is also true. By Theorem 1 there is a perfect matching in the graph.

To calculate the complexity let us perform the following observations. The number of jobs is $n = mb$. The graph constructed in step 3 has $O(b)$ vertices and $O(b^2)$ edges. It can be constructed in O(mb) time by visiting all machines $m_{i' \geq i}$. The complexity of the Hopcrof-Karp algorithm is $O((|E| + |V|)|V|^{1/2})$ (Hopcroft and Karp 1973), hence here it is $O(b^{5/2})$. The exchange can be done in $O(b)$ time. Together this gives time $O(bm + b^{5/2})$ □

Consider an instance of $P|\text{cliques}|\Sigma C_j$. Assume that each of the cliques have m jobs. If this is not the case, add jobs with processing time 0. Now there is $O(nm)$ jobs, because $b = O(n)$. Order the jobs non-increasingly according to their processing times and schedule them in a round robin fashion without respect to the cliques, which is by the Smith's Rule optimal (Smith 1956). By Lemma 1 we may easily construct a method to change the schedule to respect the cliques, by calling Algorithm 1 m times.

Theorem 2. *$P|\text{cliques}|\Sigma C_j$ can be solved to optimality in $O(nm^2 + n^{5/2}m)$ time.*

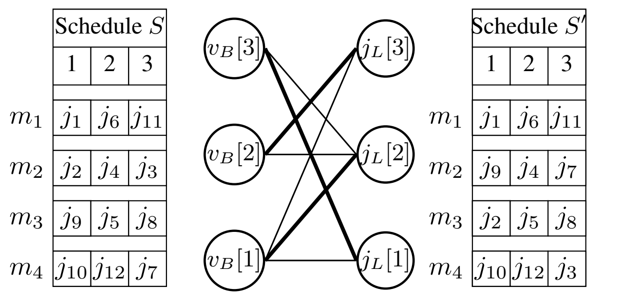

Figure 1: An illustration of an application of Algorithm 1. Let the set of cliques be given by $\{j_1, j_2, j_3, j_4\}$, $\{j_5, j_6, j_7, j_8\}$, $\{j_9, j_{10}, j_{11}, j_{12}\}$ and let $i = 2$ (which means that m_1 has already a set of compatible jobs assigned). Notice how using a matching in the constructed graph the jobs can be exchanged in a way that m_2 has only compatible jobs assigned.

Algorithm 2 An exact algorithm for $R|\text{cliques}, M(j), (p_k^i)_{k\in[b], i\in M}|\sum C_j$.

```
Require: A set of cliques $V_1 \cup \ldots \cup V_b$, a set of $m$ machines $M$, a mapping between machines and cliques, and relation $compatible \subseteq J \times M$ between jobs and machines.
Ensure: An optimal schedule
 1: Construct a flow network:
 2: Let there be sinks $T = M \times \{1, \ldots, n\}$, each with capacity 1.
 3: Let there be sources $S = V_1 \cup \ldots \cup V_b$, each with capacity 1.
 4: Let there be vertices $V^1 = M \times \{1, \ldots, b\} \times \{1\}$.
 5: Let there be vertices $V^2 = M \times \{1, \ldots, b\} \times \{2\}$.
 6: $A_1 = \{(j, (m, i, 1)) | j \in V_i, m \in M, (j, m) \in compatible\}$.
 7: $A_2 = \{((m, i, 1), (m, i, 2)) | m \in M, i \in \{1, \ldots, b\}\}$.
 8: $A_3 = \{((m, i, 2), (m, n')) | n' \in \{1, \ldots, n\}, m \in M, i \in \{1, \ldots, b\}\}$.
 9: Let $A = A_1 \cup A_2 \cup A_3$.
10: $\forall_{e\in A} capacity(e) = 1$.
11: $\forall_{e\in A} cost(e) = \left\{ \begin{array}{ll} n'p_k^i & | \, e = ((i,k,2),(i,n')) \\ 0 & | \, otherwise \end{array} \right\}$.
    {By an abuse of the notation, we assume that for a clique $V_k$, $p_k^i$ is the processing time of a job from $V_k$ on $m_i$.}
12: Construct the maximum flow with minimal cost in $(S \cup T \cup V^1 \cup V^2, A, capacity, cost)$.
13: return If the flow is less than $n$, then there is no feasible schedule. Otherwise return a schedule corresponding to the flow.
```

By constructing a suitable flow network, similar to the one used in (Bruno, Jr., and Sethi 1974), but with the cliques requirement satisfied we have:

Theorem 3. *$R|\text{cliques}, M(j), (p_k^i)_{k\in[b], i\in M}|\sum C_j$ can be solved to optimality in $O(n^4 m + n^3 m^2 \log nm)$ time.*

Proof. Consider Algorithm 2. Observe that the constructed flow network has integral capacities, hence there exist an integral flow that has minimum cost. The flow network is an adaptation of the network presented in (Bruno, Jr., and Sethi 1974). It is easy to see that a schedule corresponding to such a flow respects the cliques due to capacities of A_2. Also it respects the restrictions of the jobs by the composition of A_1. The constructed network has $O(mn)$ vertices and $O(n^2 + nm)$ arcs. The complexity follows from the size of the network and an algorithm for min-cost flow. Precisely, by an observation that in our network the maximum flow is $O(n)$. Hence, the successive shortest path algorithm has the time complexity $O(n^2 m(n^2 + nm \log nm))$ (Ahuja, Magnanti, and Orlin 1993). The part $(n^2 + nm \log nm)$ in the formula is due to the time complexity of an algorithm for shortest path problem. The Fibonacci heap implementation may be used, which has the time complexity $O(n^2 + nm + nm \log nm) = O(n^2 + nm \log nm)$. □

Hardness Results

We prove that $P|\text{cliques}, M(k), p_i \in \{p_1 < p_2 < 2p_1\}|\sum C_j$ is APX-hard by an L-reduction from the problem MAX 3SAT-6. Let us start with a description of this problem. It is an optimization version of 3SAT in which every variable appears in 6 clauses and each literal in exactly 3 clauses. The goal is to calculate the maximum number of clauses that can be satisfied, i.e., have at least one literal with truth assigned. From the sketch of the proof of Theorem 12 from (Feige, Lovász, and Tetali 2004) we get the following lemma.

Lemma 2 (Feige, Lovász, and Tetali 2004). *The problem* MAX 3SAT-6 *is* APX-hard.

We use L-reduction in the sense of (Ausiello et al. 1999) to prove the following theorem.

Theorem 4. *$P|\text{cliques}, M(k), p_i \in \{p_1 < p_2 < 2p_1\}|\sum C_j$ is* APX-hard.

Proof. For the pair of the problems let us define f, the function constructing an instance of $P|\text{cliques}, M(k), p_i \in \{p_1 < p_2 < 2p_1\}|\sum C_j$ from an instance of MAX 3SAT-6. Let the set of variables be V; and the set of clauses be $\mathcal{C}$, where $|\mathcal{C}| = 2|V|$. Define $\kappa : V \times \{1, \ldots, 6\} \to \mathcal{C} \times \{1, 2, 3\}$ to be a function that maps the first unnegated literal of a variable, the first negated literal of the variable, etc. to its clause and the position in the clause. For a variable $v \in V$, construct a set of machines $\cup_{i=1,\ldots,6}\{m[v, i]\}$. The machine $m[v, 1]$ corresponds to the first non-negated literal of v, $m[v, 2]$ corresponds to first negated one, etc. Construct also a set of clause machines $\{m[C, 1], m[C, 2], m[C, 3]\}$, for $C \in \mathcal{C}$. The jobs that we construct are described in Table 2. Notice that there are $13|V|$ jobs with size p_1 and $11|V|$ jobs with size p_2. The construction is illustrated in Fig. 2.

Let k be the maximum number of clauses that can be satisfied for a given instance of MAX 3SAT-6. Notice that $|V| \le k \le 2|V|$, because if we assign T to all the variables, then at least half of the clauses are satisfied. Let us make an assignment of the jobs to machines based on a valuation giving k satisfied clauses. Consider two cases.

- If a variable v has value T, let $m[v, 1], m[v, 3], m[v, 5]$ be assigned jobs $j[v, 1], j[v, 3], j[v, 5]$ and

job	clique	p_i	clique allowed on
$j[v,1]$	$V[v,1]$	p_1	$m[v,1], m[v,2]$
$j[v,2]$	$V[v,2]$	p_2	$m[v,2], m[v,3]$
$j[v,3]$	$V[v,3]$	p_1	$m[v,3], m[v,4]$
$j[v,4]$	$V[v,4]$	p_2	$m[v,4], m[v,5]$
$j[v,5]$	$V[v,5]$	p_1	$m[v,5], m[v,6]$
$j[v,6]$	$V[v,6]$	p_2	$m[v,6], m[v,1]$
$j^T[v,i]$	$V^*[v,i]$	p_1	$m[v,i], m[\kappa(v,i)]$
$j^F[v,i]$	$V^*[v,i]$	p_2	$m[v,i], m[\kappa(v,i)]$
$j[C,1]$	$V[C,1]$	p_1	$m[C,1], m[C,2], m[C,3]$
$j[C,2]$	$V[C,1]$	p_1	$m[C,1], m[C,2], m[C,3]$
$j[C,3]$	$V[C,1]$	p_2	$m[C,1], m[C,2], m[C,3]$

Table 2: The processing times p_i of jobs used in the L-reduction in Theorem 4.

let $m[v,2], m[v,4], m[v,6]$ be assigned jobs $j[v,2], j[v,4], j[v,6]$.

- Otherwise let $m[v,1], m[v,3], m[v,5]$ be assigned jobs $j[v,6], j[v,2], j[v,4]$ and let $m[v,2]$, $m[v,4], m[v,6]$ be assigned jobs $j[v,1], j[v,3], j[v,5]$.

If $m[v,i]$ has job with processing time p_2 assigned already, assign a job with processing time p_1 from $V^*[v,i]$ to it; otherwise assign a job with processing time p_2 from $V^*[v,i]$ to it. Assign the other job from $V^*[v,i]$ to $m[\kappa(v,i)]$. For all $C \in \mathcal{C}$ assign the jobs from the clique $V[C,1]$ to the eligible machines in an optimal way. Notice that only the machines that correspond to the clauses that are not satisfied can have two jobs with size p_2 assigned, and there is exactly one such machine for a given not satisfied clause. Notice that the cost of such a schedule is $6|V|(2p_1+p_2)+(2|V|-k)(4p_1+5p_2)+(11|V|-6|V|-4(2|V|-k))(2p_1+p_2)+\frac{1}{2}(13|V|-6|V|-2(2|V|-k)-(11|V|-6|V|-4(2|V|-k)))3p_1 = 25|V|p_1+11|V|p_2+(2|V|-k)(p_2-p_1) \leq k(24p_1+12p_2)$. Hence let $(24p_1+12p_2)$ be the β constant, bounding the value of optimal solution for MAX 3SAT-6 in terms of the value of optimal solution for the scheduling problem.

Let us assume that for a given instance of MAX 3SAT-6 we have a solution y of the corresponding instance of the scheduling problem with a given cost. Let us define the g function, a function which constructs a solution of MAX 3SAT-6 from a solution of the scheduling problem. The g function begins with modifying the solution according to the following observations.

1. Let us assume that in y there exists $m[v,i]$ that has exactly 2 jobs assigned; let us assume that both of them have size p_1 (have size p_2). Notice that this means that the machine has a job $j^T[v,i]$ (a job $j^F[v,i]$) assigned. Notice that we can exchange this job with $j^F[v,i]$ (with $j^T[v,i]$) without increasing the total completion time.
2. Assume that some machine $m[v,i]$ has three jobs assigned. It also means that there is a machine $m[v,i']$ that has exactly one job assigned. Notice that in any case, by the previous observation and by the assumption that $p_1 \leq p_2 \leq 2p_1$ we may shift the jobs in a way that after the shift all of the machines have exactly 2 jobs, without increasing the total completion time of the schedule. This follows from a simple analysis of all possible cases of the assignment of the jobs to the machines (see (Jansen et al. 2020) for the sketch of analysis).

Notice that this means that we may assume that the machines $m[v,i]$ are processing exactly one job with size p_1 and one with size p_2 each. We prove that the total completion time of the schedule depends only on the number of the machines that are processing two jobs with size p_2. Let the number of such machines be k'. Total completion time of the schedule is then equal to $k'3p_2+(11|V|-2k')(2p_1+p_2)+\frac{1}{2}(13|V|-(11|V|-2k'))3p_1 = 25|V|p_1+11|V|p_2+k'(p_2-p_1)$. From such a schedule we can easily find a valuation of the variables in the instance of MAX 3SAT-6 such that it satisfies exactly $2|V|-k'$ clauses. Let now k'' be the number of machines that are processing two jobs with size p_2 in an optimal solution. Notice that k'' corresponds to a schedule with

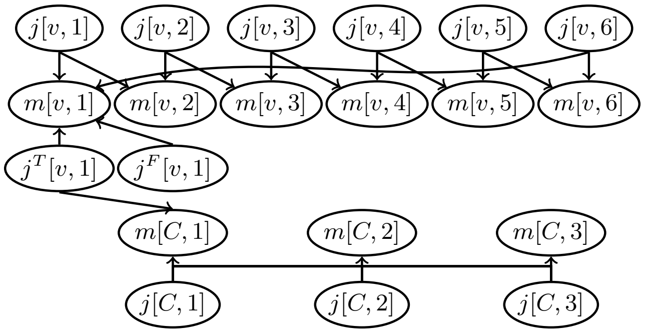

Figure 2: An illustration of the idea of eligibility of the jobs used in Theorem 4. The figure presents a component corresponding to one of the variables and a component corresponding to one of the clauses. In the example C is such a clause that $(C,1) = \kappa(v,1)$.

cost $25|V|p_1+11|V|p_2+k''(p_2-p_1)$. And this schedule corresponds to a solution to MAX 3SAT-6 that has exactly $(2|V|-k'')$ clauses satisfied. There can be no better solution to MAX 3SAT-6. Hence let us assume that for some γ we have that $|(2|V|-k'')-(2|V|-k')| \leq \gamma|k''(p_2-p_1)+25|V|p_1+11|V|p_2-(k'(p_2-p_1)+25|V|p_1+11|V|p_2)|$. Which is equivalent to $k'-k'' \leq \gamma(k'-k'')(p_2-p_1)$. Hence clearly $\gamma = \frac{1}{p_2-p_1}$ is a suitable constant, bounding the difference of values of an optimal solution for a given instance of MAX 3SAT-6 and a solution obtained by a transformation of the solution of the prepared scheduling instance; with the difference between values of an optimal solution for the instance of the scheduling problem and any (reasonable) solution to the instance of the scheduling problem. All other conditions are easily fulfilled. $\square$

The APX-hardness for $R|\text{cliques}, p_j^i \in \{p_1 < p_2 < p_3\}|\sum C_j$ follows readily from the observation that we may always set p_3 to such a high value (dependent on the size of an instance - mind the difference with the previous problem) that in any reasonable schedule it will be not used.

The presented idea partially follows from the next problem and an α-reduction, this time from an even more restricted version, i.e., from the problem 3SAT* considered in (Maack and Jansen 2020). The input of 3SAT* problem consists of a set of variables, and two sets of clauses: 1-in-3 clauses and 2-in-3 clauses. Each of the literals occurs exactly 2 times, hence each variable occurs exactly twice negated and twice nonnegated. The number of 1-in-3 clauses and 2-in-3 clauses are equal. The question is if there is assignment of the variables such that in each 1-in-3 clause exactly one literal is true and that in each 2-in-3 clause exactly two literals are true. In the paper it was proved that the problem is NP-complete. For the next problem we use p_3 to restrict assignment of some jobs to some machines. We have to also divide the jobs differently.

Theorem 5. *$R|2\text{ cliques}, p_j^i \in \{p_1 < p_2 < p_3\}|\sum C_j$ is* NP-complete.

FPT Results

This section presents the FPT results for scheduling with clique incompatibility considering different parameterizations. To solve these problems, the algorithms model the respective problem as n-fold Integer Programs. These IPs are of specific form. The constraint matrix consists of non-zero entries only in the first few rows and in blocks along the diagonal beneath. Further, we have to assure that the introduced objective functions are separable convex. Then the n-fold IP and thus, the underlying problem can be solved efficiently. The FPT results we obtain this way are:

- $P|\text{cliques}, M(k)|\sum C_j$ can be solved in FPT time parameterized by the number of cliques b,
- $R|\text{cliques}|\sum w_j C_j$ can be solved in FPT time parameterized by the number of machines m, the largest processing time $p_{\max}$, and the number of job kinds ϑ.

The basis for the last algorithm is formed by the work (Knop and Koutecký 2018). Therein, the authors prove FPT results for $R||\sum w_j C_j$ by formulating the problems as n-fold IPs with an appropriate objective function and similar parameters. We prove that the IP can be extended to handle clique incompatibility by carefully adapting the variables, the IP and the objective function, yielding the result above. Note that in (Knop et al. 2019), these results are generalized, but by that also more complex. Further, using these results does not improve upon our running times.

But first, let us give a short introduction to FPT and n-fold Integer Programming necessary to understand the results. For details on FPT, we refer to the standard textbook (Cygan et al. 2015). For details on n-fold IPs, we recommend (Eisenbrand et al. 2019).

FPT. We say that an optimization problem is *fixed-parameter tractable* (FPT) with respect to some instance-dependent parameter k if there exists an algorithm which, for an instance of size n, finds an optimal solution in time $f(k) \cdot n^{O(1)}$, where f is some function depending only on k (Mnich and Wiese 2015).

n-fold IP. Let $n, r, s, t \in \mathbb{N}$. Let $A_1, \dots, A_n \in \mathbb{Z}^{r \times t}$ and $B_1, \dots, B_n \in \mathbb{Z}^{s \times t}$ be integer matrices. The constraint matrix $\mathcal{A} \in \mathbb{Z}^{(r+n\cdot s)\times(n\cdot t)}$ of an n-fold IP is of following form:

$$\mathcal{A} = \begin{pmatrix} A_1 & A_2 & \dots & A_n \\ B_1 & 0 & \dots & 0 \\ 0 & B_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & B_n \end{pmatrix}.$$

Denote by Δ the largest absolute value in $\mathcal{A}$. We distinguish the constraints as follows: Denote the constraints (rows) corresponding to the A_i matrices *globally uniform* and the ones corresponding to the B_i matrices *locally uniform*.

A function $g : \mathbb{R}^n \to \mathbb{R}$ is called separable convex if there exist convex functions $g_i : \mathbb{R} \to \mathbb{R}$ for each $i \in [n]$ such that $g(x) = \sum_{i=1}^{n} g_i(x_i)$. Let $f : \mathbb{R}^{nt} \to \mathbb{R}$ be some separable convex function and $b \in \mathbb{Z}^{r+n\cdot s}$. Further, denote by ℓ and u some lower and upper bounds on the variables. The corresponding n-fold Integer Program (n-fold IP) is defined by $\min\{f(x) \mid \mathcal{A}x = b, \ell \leq x \leq u, x \in \mathbb{Z}^{n\cdot t}\}$. The main idea for solving these IPs relies on local improvement steps which are used to converge from an initial solution to an optimal one yielding:

Proposition 1 (Eisenbrand et al. 2019). *The Integer Program (n-fold IP) can be solved in time $(\Delta rs)^{O(r^2 s + rs^2)}$ $nt\log(nt)\log(\|u - \ell\|_\infty)\log(f_{\max})$ where $f_{\max} = \max\{|f(x)| \mid \ell \leq x \leq u\}$.*

Scheduling with Clique Machine Restrictions

We consider $P|\text{cliques}, M(k)|\sum C_j$. Recall that in this setting, we have a set $M(k)$ of machines for each clique $k \in [b]$. In a feasible schedule jobs of k are scheduled exclusively on machines $i \in M(k)$. We prove the following result:

Theorem 6. *$P|\text{cliques}, M(k)|\sum C_j$ can be solved in FPT time parameterized by the number of cliques b.*

To prove this result, we first establish some notation and basic observation, then introduce an Integer Programming model with n-fold form for the problem, and lastly argue that it can be solved efficiently.

In any schedule for an instance of the problem there can be at most b jobs scheduled on each machine. Hence, we may imagine that there are b slots on each machine numbered in chronological order. We further use the discussed intuition that the jobs form b layers. We can represent any schedule by an assignment of the jobs to these slots. Some of the slots may be empty, and we introduce the convention that all the empty slots (hence taking 0 time) on a machine should be in the beginning. If a job of clique k is scheduled in a certain slot, we say that k is present in the slot, in the corresponding layer and on the corresponding machine. We are interested in the pattern of cliques present on the machine and call such a pattern a *configuration*. More precisely, we call a vector $C = (C_1, C_2, \dots, C_b) \in \{0, 1, \dots, b\}^b$ a configuration if the following two conditions are satisfied:

- $\forall \ell, \ell' \in [b] : C_\ell = C_{\ell'} \wedge \ell \neq \ell' \implies C_\ell = C_{\ell'} = 0$
- $\forall \ell \in [b-1] : C_\ell > 0 \wedge \ell < b \implies C_{\ell+1} > 0$

Note that the 0 represents an empty slot. The first condition corresponds to the requirement that at most one job of a clique should be scheduled on each machine. The second one matches to the convention that the empty slots are at the beginning. We denote the set of configurations as $\mathcal{C}$. Moreover, $\mathcal{C}(k)$ denotes for each $k \in [b]$ the set of configurations in which k is present, i.e., $\mathcal{C}(k) = \{C \in \mathcal{C} \mid \exists \ell \in [b] : C_\ell = k\}$. Note that $|\mathcal{C}| \leq (b+1)!$ since there can be up to b zeros in a configuration and a configuration excluding the zeros can be seen as a truncated permutation of the numbers in $[b]$. We call a configuration C eligible for a machine i if all the cliques occurring in C are eligible on i, that is, for each $C_\ell \neq 0$ we have $i \in M(C_\ell)$.

A schedule for an instance of the problem trivially induces an assignment of the machines to the configurations. We call such an assignment $\tau : M \to \mathcal{C}$ feasible if there exists a feasible schedule corresponding to τ. That is, if $\tau(i)$ is eligible on i for each machine i and, for each clique k, the number of machines assigned to a configuration in $\mathcal{C}(k)$ is equal to the number of jobs in k. Obviously, different schedules may have the same assignment. However, we argue that given a feasible assignment τ, we can find a schedule corresponding to τ with a minimal objective function value via a simple greedy procedure. Namely, for each clique k we can successively choose a smallest job that is not scheduled yet and assign it to a slot positioned in the lowest layer that still includes non-empty slots belonging to k according to τ. Due to this observation, we can associate an objective value to each feasible assignment. In the next step we introduce an Integer Program to search for a feasible assignment τ with minimal objective.

We introduce two types of variables, that is, $x_{C,i} \in \{0,1\}$ for each machine $i \in M$ and configuration $C \in \mathcal{C}$ corresponding to the choice of whether i is assigned to C or not. Further, we have $y_{k,\ell} \in \{0,1,\dots,n\}$ for each clique $k \in [b]$ and layer $\ell \in [b]$ counting the number slots reserved for clique k in the layers 1 to ℓ. Moreover, we assure $x_{C,i} = 0$ if C is not eligible on i using more restrictive upper bounds. Let $\mathcal{C}(k,\ell) = \{C \in \mathcal{C} \,|\, \exists \ell' \in [\ell] : C_{\ell'} = k\}$ for each $k,\ell \in [b]$, n_k be the number of jobs belonging to clique k, and $p_{k,s}$ the size of the job that has position s if we order the jobs of clique k non-decreasingly by size. The Integer Program has the following form:

$$\min \sum_{\ell,k\in[b]} \sum_{s=1}^{y_{k,\ell}} p_{k,s}$$

$$\sum_{C\in\mathcal{C}} x_{C,i} = 1 \qquad \forall i \in M \qquad (1)$$

$$\sum_{i\in M} \sum_{C\in\mathcal{C}(k,\ell)} x_{C,i} = y_{k,\ell} \qquad \forall k \in [b], \ell \in [b] \qquad (2)$$

$$y_{k,b} = n_k \qquad \forall k \in [b] \qquad (3)$$

Constraint (1) assures that exactly one configuration is chosen for each machine; due to (2), the variables $y_{k,\ell}$ correctly count the slots reserved for clique k; and (3) guarantees that the jobs of each clique are covered. Finally, the objective function corresponds to the one described above: For each clique k, we sum up the smallest $y_{k,1}$ job sizes for the first layer, the smallest $y_{k,2}$ sizes in the second one, and so on. Note that this counting is correct since we use the convention that empty slots are at the bottom and therefore, each job contributes once to the objective for its own layer and once for each layer above. Although the integer program does not have a linear objective and super-constant number of variables and constraints, we can solve it in suitable time using n-fold techniques:

Lemma 3. *The above IP can be solved in time* $2^{O(b^4 \cdot \log(b))} m \log(m) \log(n) \log(m p_{\max})$.

Proof. In order to use algorithms for n-fold IPs, we have to show that the IP has a suitable structure and the objective function is separable convex.

To obtain the desired structure, we have to duplicate the y variables for each machine. Hence, we get variables $y_{k,\ell,i}$ for each $i \in M$ and $k,\ell \in [b]$. We choose some machine $i^* \in M$ and set $y_{k,\ell,i} = 0$ for each $i \neq i^*$ using lower and upper bounds for the variables. In the constraints (2) and (3) we have to replace each occurrence of $y_{k,\ell}$ by $\sum_{i\in M} y_{k,\ell,i}$. Moreover, we have to change the objective to $\min \sum_{\ell,k\in[b]} \sum_{s=1}^{y_{k,\ell,i^*}} p_{k,s}$. It is easy to see that the resulting IP is equivalent and has an n-fold structure with one brick for each machine, a brick size of $t \le b^2 + (b+1)!$, and a maximum absolute entry of $\Delta = 1$. Constraint (1) is locally uniform, and the other constraints are globally uniform. Hence, we have $s = 1$ and $r = b^2 + b$.

Concerning the objective function, first note that many of the variables do not occur in the objective and hence, can be ignored in the following. We essentially have to consider the function $g_k : [n_k] \to \mathbb{R}, q \mapsto \sum_{s=1}^{q} p_{k,s}$ for each $k \in [b]$ since the objective can be written as $\sum_{\ell,k\in[b]} g_k(y_{k,\ell,i^*})$. Let $\{x\} = x - \lfloor x \rfloor$ for each $x \in \mathbb{R}$ and $\tilde{g}_k : \mathbb{R} \to \mathbb{R}$ with:

$$x \mapsto \begin{cases} p_{k,1}x & \text{if } x < 1 \\ p_{k,\lceil x \rceil}\{x\} + \sum_{s=1}^{\lfloor x \rfloor} p_{k,s} & \text{if } \lfloor x \rfloor \in [n_k - 1] \\ p_{k,n_k}(x - n_k) + \sum_{s=1}^{n_k} p_{k,s} & \text{if } x \ge n_k \end{cases}.$$

Then we have $\tilde{g}_k(q) = g_k(q)$ for each $k \in [n_k]$. Furthermore, $\tilde{g}_k$ is continuous and essentially a linear function with $n_k - 1$ points at which the slope changes. Due to the ordering of the processing times, the slope can only increase and hence, the function is convex.

Finally, note that maximal value $f_{\max}$ of the objective function can be upper bounded by $p_{\max} b^2 n$ and the maximal difference between the upper and lower bound of a variable is given by n. By plugging in the proper values, Proposition 1 yields the stated running time. ☐

Scheduling with Cliques for Sum of Weighted Completion Times

We consider here $R|\text{cliques}|\sum w_j C_j$. Recall that we are given m machines forming a set M and n jobs forming a set J. Each job $j \in J$ has an m-dimensional vector $p_j = (p_j^1, \dots, p_j^m) \in \mathbb{Z} \cup \{\infty\}$ stating that job j has a processing time p_j^i on machine $i \in M$. Also, each job has a weight w_j. The jobs are partitioned into b cliques. Further, we introduce *kinds* of jobs formally. Two jobs belong to the same kind if their processing time vectors are equal and their weights are the same. Denote the number of *job kinds* as ϑ. We can re-write the set of jobs as $(n_{1,1}, \dots, n_{\vartheta,1}, n_{1,2}, \dots, n_{\vartheta,b})$ where jobs of kind j and clique k appear $n_{j,k}$ times. Denote by $p_{\max}$ the largest processing time and by $w_{\max}$ the largest weight occurring in the instance. In the remaining of this section, we prove the following theorem:

Theorem 7. $R|\text{cliques}|\sum w_j C_j$ *can be solved in FPT time parameterized by the number of machines* m*, the largest processing time* $p_{\max}$ *and the number of job kinds* ϑ*.*

The main obstacle in the design of an n-fold IP for this setting is to formulate an appropriate objective function. In (Knop and Koutecký 2018) it was developed a quadratic separable convex function equivalent to $\sum w_j C_j$. This result relies on the fact that in an optimal schedule the jobs on each machine are ordered regarding the Smith's rule, i.e., non-increasingly regarding $\rho_i(j) = w_j/p_j^i$ (Goemans and Williamson 2000). We may visualize this as a Gantt chart for each machine: Roughly speaking, it is a line of neighboring rectangles in the order of the schedule. The width of the i-th rectangle is the processing time of the ith job on the machine and the rectangles height corresponds to the total weight of all uncompleted jobs (including the ith job). The area under the function, i.e. an integral of the weights of uncompleted jobs in time, corresponds to the weighted completion time and can be separated into two parts. One part is dependent only on the job kind and machine kind. The second one is dependent on the composition of the jobs assigned to the machine. By the fact that for any machine, the Smith's order is optimal, the order of job kinds is known. Hence, the composition is determined by the number of jobs of each kind assigned to the machine. Thus, the second part yields a piece-wise linear convex function. For details see (Knop and Koutecký 2018). This gives:

Proposition 2 (Knop and Koutecký 2018). *Let $x_1^i, \ldots, x_\vartheta^i$ be numbers of jobs of each kind scheduled on a machine m_i and let $\pi_i\colon [\{1,\ldots,\vartheta\}] \to [\{1,\ldots,\vartheta\}]$ be a permutation of job kinds such that $\rho_i(\pi_i(j)) \geq \rho_i(\pi_i(j+1))$ for all $1 \leq j \leq \vartheta - 1$. Then the contribution of m_i to the weighted completion time in an optimal schedule is equal to $\sum_{j=1}^{\vartheta}(1/2(z_j^i)^2(\rho_i(\pi_i(j)) - \rho_i(\pi_i(j+1))) + 1/2 \cdot x_j^i p_j^i w_j)$ where $z_j^i = \sum_{\ell=1}^{j} p_{\pi_i(\ell)}^i x_{\pi_i(\ell)}^i$.*

Proof. First, let us focus on constructing the n-fold IP. For this result, we extend the n-fold IP introduced in (Knop and Koutecký 2018) and adapt the separable convex function to our needs. Let $x_{j,k}^i$ be a variable that corresponds to the number of jobs of kind $j \in \{1,\ldots,\vartheta\}$ from clique $k \in \{1,\ldots,b\}$ that are scheduled on machine $i \in \{1,\ldots,m\}$. Consider the following IP:

$$\sum_{k=1}^{b}\sum_{\ell=1}^{j} x_{\pi_i(\ell),k}^i p_{\pi_i(\ell)}^i = z_j^i \qquad \forall j \in [\vartheta], \forall i \in [m] \qquad (1)$$

$$\sum_{i=1}^{m} x_{j,k}^i = n_{j,k} \qquad \forall j \in [\vartheta], \forall k \in [b] \qquad (2)$$

$$\sum_{j=1}^{\vartheta} x_{j,k}^i \leq 1 \qquad \forall i \in [m], \forall k \in [b] \qquad (3)$$

with lower bounds 0 for all variables and upper bounds $x_{j,k}^i \leq 1$ and $z_j^i \leq b \cdot p_{\max}$.

Let the $x_{j,k}^i$ variables form a vector $\mathbf{x}$ and the z_j^i variables from a vector $\mathbf{z}$. Denote by $\mathbf{x}^i$ and $\mathbf{z}^i$ the corresponding subset restricted to one machine i. The objective is to minimize the function $f(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{m} f^i(\mathbf{x}^i,\mathbf{z}^i) = \sum_{j=1}^{\vartheta}(1/2(z_j^i)^2(\rho_i(\pi_i(j)) - \rho_i(\pi_i(j+1))) + 1/2 \cdot \sum_{k=1}^{b} x_{j,k}^i p_j^i w_j)$. As we consider the altered variables $x_{j,k}^i$ over all cliques simultaneously, this corresponds to the objective function from Proposition 2. Thus, the function expresses the sum of completion times objective. Further it obviously stays separable convex.

Regarding the constraint matrix, Constraint (1) is satisfied if the z_j^i variables are set as demanded in Proposition 2, i.e., the jobs are scheduled with respect to the Smith's rule. Constraint (2) assures that the number of jobs from a kind j and clique k scheduled on the machines matches the overall number of jobs from that kind and clique. Finally, Constraint (3) assures that the number of jobs scheduled on a machine i from the same clique k is at most one.

We construct a schedule from the solution of the above IP as follows: Assign $x_{j,k}^i$ jobs of job kind j from clique k to machine i (note that this number is at most one due to Constraint (3)). After assigning all jobs to a machine, place them non-increasingly regarding the Smith's ratio $\rho_i(j)$ onto the machine. As we did not change the objective from (Knop and Koutecký 2018), such a solution corresponds to an optimal one regarding the sum of weighted completion times objective.

Regarding the running time, we first have to estimate the n-fold IP parameters. The first constraint is globally uniform whereas the second and third constraints are locally uniform and repeated for each clique. The parameters can be bounded by

$n = b, t = O(\vartheta \cdot m), r = O(\vartheta \cdot m), s = O(m + \vartheta)$, $\Delta = p_{\max}, \log(||u - \ell||_\infty) = \log(b \cdot p_{max})$, $\log(f_{max}) = O(\log(m \cdot b^2 \cdot p_{\max} \cdot w_{\max})) \leq O(\log(m \cdot b \cdot p_{\max} \cdot w_{\max}))$.

Applying Proposition 1 to solve the IP and setting up the schedule results in a running time of $(p_{\max}\vartheta m)^{O(\vartheta^3 m^3)} O(b \log^3(b) \log(w_{\max}))$. Note that the inequality constraints do no harm as we can introduce parameter many slack-variables to turn them into equality constraints. $\square$

Consider $R|\text{cliques}|\sum w_j C_j$ but parameterized by b, $p_{\max}$ and ϑ. The n-fold IP solving the following problem is an extended formulation of the one from (Knop and Koutecký 2018), but with additional consideration for cliques - we have to embed them appropriately. It proves,

Theorem 8. *$R|\text{cliques}|\sum w_j C_j$ can be solved in FPT time parameterized by the number of cliques b, the largest processing time $p_{\max}$ and the number of job kinds ϑ, in a running time of $(b\vartheta p_{\max})^{O(\vartheta^3 b^3)} O(m \log^2(m) \log(w_{\max}))$.*

Open Problems

While this paper presents quite a few results, many research directions are still open. For instance, the case of uniformly related machines $Q|\text{cliques}|\sum C_j$, where the machines have different speeds is more general than $P|\text{cliques}|\sum C_j$, but in turn more restricted than $R|\text{cliques}|\sum C_j$, remains open. Another question is whether a constant rate approximation algorithm for $P|\text{cliques}, M(k)|\sum C_j$ is possible, given that it is APX-hard. Furthermore, we are interested in a more detailed study of our setting from the perspective of approximation algorithms or even approximation algorithms with FPT running times. Finally, the study of further sensible classes of incompatibility graphs for $\sum C_j$ objective seems worthwhile.

Acknowledgments

Supported by DFG Projects "Strukturaussagen und deren Anwendung in Scheduling- und Packungsprobleme", JA 612/20-1 and, JA 612/20-2; and by Gdańsk University of Technology, grant no. POWR.03.02.00-IP.08-00-DOK/16.

References

Ahuja, R. K.; Magnanti, T. L.; and Orlin, J. B. 1993. *Network Flows - Theory, Algorithms and Applications*. Prentice Hall. ISBN 978-0-13-617549-0.

Ausiello, G.; Marchetti-Spaccamela, A.; Crescenzi, P.; Gambosi, G.; Protasi, M.; and Kann, V. 1999. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer. ISBN 3540654313.

Bodlaender, H. L.; and Jansen, K. 1993. On the Complexity of Scheduling Incompatible Jobs with Unit-Times. In *MFCS*, volume 711 of *Lecture Notes in Computer Science*, 291–300. Springer.

Bodlaender, H. L.; Jansen, K.; and Woeginger, G. J. 1994. Scheduling with Incompatible Jobs. *Discret. Appl. Math.* 55(3): 219–232.

Brucker, P. 2004. *Scheduling algorithms (4. ed.)*. Springer. ISBN 978-3-540-20524-1.

Bruno, J. L.; Jr., E. G. C.; and Sethi, R. 1974. Scheduling Independent Tasks to Reduce Mean Finishing Time. *Commun. ACM* 17(7): 382–387.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company. ISBN 0-262-03293-7.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

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

Dokka, T.; Kouvela, A.; and Spieksma, F. C. R. 2012. Approximating the Multi-Level Bottleneck Assignment Problem. *Oper. Res. Lett.* 40(4): 282–286.

Eisenbrand, F.; Hunkenschröder, C.; Klein, K.; Koutecký, M.; Levin, A.; and Onn, S. 2019. An Algorithmic Theory of Integer Programming. *CoRR* abs/1904.01361.

Feige, U.; Lovász, L.; and Tetali, P. 2004. Approximating Min Sum Set Cover. *Algorithmica* 40(4): 219–234.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. ISBN 0-7167-1044-7.

Goemans, M. X.; and Williamson, D. P. 2000. Two-Dimensional Gantt Charts and a Scheduling Algorithm of Lawler. *SIAM J. Discret. Math.* 13(3): 281–294.

Grage, K.; Jansen, K.; and Klein, K. 2019. An EPTAS for Machine Scheduling with Bag-Constraints. In *SPAA*, 135–144. ACM.

Hall, P. 1935. On Representatives of Subsets. *Journal of the London Mathematical Society* s1-10(1): 26–30.

Hopcroft, J. E.; and Karp, R. M. 1973. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.* 2(4): 225–231.

Jansen, K.; Lassota, A.; Maack, M.; and Pikies, T. 2020. Total Completion Time Minimization for Scheduling with Incompatibility Cliques. *CoRR* abs/2011.06150.

Knop, D.; and Koutecký, M. 2018. Scheduling Meets n-Fold Integer Programming. *J. Sched.* 21(5): 493–503.

Knop, D.; Koutecký, M.; Levin, A.; Mnich, M.; and Onn, S. 2019. Multitype Integer Monoid Optimization and Applications. *CoRR* abs/1909.07326.

Maack, M.; and Jansen, K. 2020. Inapproximability Results for Scheduling with Interval and Resource Restrictions. In *STACS*, volume 154 of *LIPIcs*, 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Mnich, M.; and van Bevern, R. 2018. Parameterized Complexity of Machine Scheduling: 15 Open Problems. *Comput. Oper. Res.* 100: 254–261.

Mnich, M.; and Wiese, A. 2015. Scheduling and Fixed-Parameter Tractability. *Math. Program.* 154(1-2): 533–562.

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

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