arXiv:1701.07242v1 [cs.DS] 25 Jan 2017

Structural Parameters for Scheduling with Assignment Restrictions*

Klaus Jansen[1], Marten Maack[1], and Roberto Solis-Oba[2]

[1]University of Kiel, Kiel, Germany, {kj,mmaa}@informatik.uni-kiel.de
[2]Western University, London, Canada, solis@csd.uwo.ca

October 8, 2018

Abstract

We consider scheduling on identical and unrelated parallel machines with job assignment restrictions. These problems are NP-hard and they do not admit polynomial time approximation algorithms with approximation ratios smaller than 1.5 unless P=NP. However, if we impose limitations on the set of machines that can process a job, the problem sometimes becomes easier in the sense that algorithms with approximation ratios better than 1.5 exist. We introduce three graphs, based on the assignment restrictions and study the computational complexity of the scheduling problem with respect to structural properties of these graphs, in particular their tree- and rankwidth. We identify cases that admit polynomial time approximation schemes or FPT algorithms, generalizing and extending previous results in this area.

1 Introduction

We consider the problem of makespan minimization for scheduling on unrelated parallel machines. In this problem a set $\mathcal{J}$ of n jobs has to be assigned to a set $\mathcal{M}$ of m machines via a schedule $\sigma : \mathcal{J} \to \mathcal{M}$. A job j has a processing time p_{ij} for every machine i and the goal is to minimize the makespan $C_{\max}(\sigma) = \max_i \sum_{j \in \sigma^{-1}(i)} p_{ij}$. In the three-field notation this problem is denoted by $R||C_{\max}$. On some machines a job might have a very high, or even infinite processing time, so it should never be processed on these machines. This amounts to assignment restrictions in which for every job j there is a subset $M(j)$ of machines on which it may be processed. An important special case of $R||C_{\max}$ is given if the machines are identical in the sense that each job j has the same processing time p_j on all the machines on which it may be processed, i.e., $p_{ij} \in \{p_j, \infty\}$. This problem is sometimes called restricted assignment and is denoted as $P|M(j)|C_{\max}$ in the three-field notation.

We study versions of $R||C_{\max}$ and $P|M(j)|C_{\max}$ where the restrictions are in some sense well structured. In particular we consider three different graphs that are defined based

*This work was partially supported by the DAAD (Deutscher Akademischer Austauschdienst) and by the German Research Foundation (DFG) project JA 612/15-1.

on the job assignment restrictions and study how structural properties of these graphs affect the computational complexity of the corresponding scheduling problems. We briefly describe the graphs. In the *primal graph* the vertices are the jobs and two vertices are connected by an edge, iff there is a machine on which both of the jobs can be processed. In the *dual graph*, on the other hand, the machines are vertices and two of them are adjacent, iff there is a job that can be processed by both machines. Lastly we consider the *incidence graph*. This is a bipartite graph and both the jobs and machines are vertices. A job j is adjacent to a machine i, if $i \in M(j)$. In Figure 1 an example of each graph is given. These graphs have also been studied in the context of constraint satisfaction (see e.g. [22] or [20]) and we adapted them for machine scheduling.

We consider the above scheduling problems in the contexts of parameterized and approximation algorithms. For $\alpha > 1$ an *α-approximation* for a minimization problem computes a solution of value $A(I) \leq \alpha \mathrm{OPT}(I)$, where $\mathrm{OPT}(I)$ is the optimal value for a given instance I. A family of algorithms consisting of $(1+\varepsilon)$-approximations for each $\varepsilon > 0$ with running times polynomial in the input length (and $1/\varepsilon$) is called a *(fully) polynomial time approximation scheme* (F)PTAS. Let π be some parameter defined for a given problem, and let $\pi(I)$ be its value for instance I. The problem is said to be *fixed-parameter tractable* (FPT) for π, if there is an algorithm that given I and $\pi(I) = k$ solves I in time $\mathcal{O}(f(k)|I|^c)$, where c is a constant, f any computable function and $|I|$ the input length. This definition can easily be extended to multiple parameters.

Related work. In 1990 Lenstra, Shmoys and Tardos [15] showed, in a seminal work, that there is a 2-approximation for $R||C_{\max}$ and that the problem cannot be approximated with a ratio better than 1.5 unless P=NP. Both bounds also hold for $P|M(j)|C_{\max}$ and have not been substantially improved since that time. The case where the number of machines is constant is weakly NP-hard and there is an FPTAS for this case [11]. In 2012 Svensson [21] presented an interesting result for $P|M(j)|C_{\max}$: A special case of the restricted assignment problem called graph balancing was studied by Ebenlendr et al. [6]. In this variant each job can be processed by at most 2 machines and therefore an instance can be seen as a (multi-)graph where the machines are vertices and the jobs edges. They presented a 1.75 approximation for this problem and also showed that the 1.5 inapproximability result remains true. Lee et al. [14] studied the version of graph balancing where (in our notation) the dual graph is a tree and showed that there is an FPTAS for it. Moreover, the special case of graph balancing where the graph is simple has been considered. For this problem Asahiro et al. [2] presented among other things a pseudo-polynomial time algorithm for the case of graphs with bounded treewidth. For certain cases of $P|M(j)|C_{\max}$ with job assignment restrictions that are in some sense well-structured PTAS results are known. In particular for the *path- and tree-hierarchical* cases ([18] and [7]) in which the machines can be arranged in a path or tree and the jobs can only be processed on subpaths starting at the leftmost machine or at the root machine respectively, and the *nested* case ([17]), where $M(j) \subseteq M(j')$, $M(j') \subseteq M(j)$ or $M(j) \cap M(j') = \emptyset$ holds for each pair of jobs (j, j').

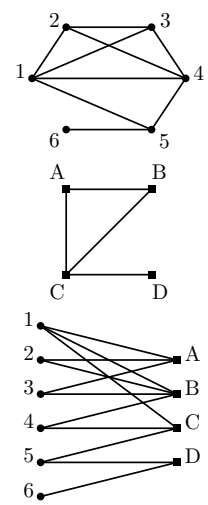

Figure 1: Primal, dual and incidence graph for an instance with 6 jobs and 4 machines.

The study of $R||C_{\max}$ from the FPT perspective has started only recently. Mnich and Wiese [16] showed that $R||C_{\max}$ is FPT for the pair of parameters m and the number of distinct processing times. The problem is also FPT for the parameter pair $\max p_{ij}$ and the number of machine types [12]. Two machines have the same type, if each job has the same processing time on them. Furthermore Szeider [23] showed that graph balancing on simple graphs with unary encoding of the processing times is not FPT for the parameter treewidth under usual complexity assumptions.

Results. In this paper we present a graph theoretical viewpoint for the study of scheduling problems with job assignment restrictions that we believe to be of independent interest. Using this approach we identify structural properties for which the problems admit approximation schemes or FPT algorithms, generalizing and extending previous results in this area. The results are based on dynamic programming utilizing tree and branch decompositions of the respective graphs. For the approximation schemes the dynamic programs are combined with suitable rounding approaches.

Tree and branch decompositions are associated with certain structural *width parameters*. We consider two of them: treewidth and rankwidth. In the following we denote the treewidth of the primal, dual and incidence graph with $\mathrm{tw_p}$, $\mathrm{tw_d}$ and $\mathrm{tw_i}$, respectively. For the definitions of these concepts we refer to Section 2.

We now describe our results in more detail. Let $J(i)$ be the set of jobs the machine i can process. In the context of parameterized algorithms we show the following.

Theorem 1. *$R||C_{\max}$ is FPT for the parameter $\mathrm{tw_p}$.*

Theorem 2. *$R||C_{\max}$ is FPT for the pair of parameters k_1, k_2 with $k_1 \in \{\mathrm{tw_d}, \mathrm{tw_i}\}$ and $k_2 \in \{\mathrm{OPT}, \max_i |J(i)|\}$.*

Note that $R||C_{\max}$ with constant k_2 remains NP-hard [6]. In the context of approximation we get:

Theorem 3. *$R||C_{\max}$ is weakly NP-hard, if $\mathrm{tw_d}$ or $\mathrm{tw_i}$ is constant and there is an FPTAS for both of these cases.*

The hardness is due to the hardness of scheduling on two identical parallel machines $P2||C_{\max}$. The result for the dual graph is a generalization of the result in [14] and resolves cases that were marked as open in that paper. All results mentioned so far are discussed in Section 3. In the following section we consider the rankwidth:

Theorem 4. *There is a PTAS for instances of $P|M(j)|C_{\max}$ where the rankwidth of the incidence graph is bounded by a constant.*

It can be shown that instances of $P|M(j)|C_{\max}$ with path- or tree-hierarchical or nested restrictions are special cases of the case when the incidence graph is a bicograph. Bicographs are known to have a rankwidth of at most 4 (see [9]) and a suitable branch decomposition can be found very easily. Therefore we generalize and unify the known PTAS results for $P|M(j)|C_{\max}$ with structured job assignment restrictions.

2 Preliminaries

In the following I will always denote an instance of $R||C_{\max}$ or $P|M(j)|C_{\max}$ and most of the time we will assume that it is feasible. We call an instance feasible if $M(j) \neq \emptyset$ for every job $j \in \mathcal{J}$. A schedule is feasible if $\sigma(j) \in M(j)$. For a subset $J \subseteq \mathcal{J}$ of jobs and a subset $M \subseteq \mathcal{M}$ of machines we denote the subinstance of I induced by J and M with $I[J, M]$. Furthermore, for a set S of schedules for I we let $\mathrm{OPT}(S) = \min_{\sigma \in S} C_{\max}(\sigma)$, and $\mathrm{OPT}(I) = \mathrm{OPT}(S)$ if S is the set of all schedules for I. We will sometimes use $\mathrm{OPT}(\emptyset) = \infty$. Note that there are no schedules for instances without machines. On the other hand, if I is an instance without jobs, we consider the empty function a feasible schedule (with makespan 0), and have therefore $\mathrm{OPT}(I) = 0$ in that case.

Dynamic programs for $R||C_{\max}$. We sketch two basic dynamic programs that will be needed as subprocedures in the following. The first one is based on iterating through the machines. Let $\mathrm{OPT}(i, J) = \mathrm{OPT}(I[\mathcal{J} \setminus J, [i]])$ for $J \subseteq \mathcal{J}$ and $i \in [m] := \{1, \ldots, m\}$, assuming $\mathcal{M} = [m]$. Then it is easy to see that $\mathrm{OPT}(i, J) = \min_{J \subseteq J' \subseteq \mathcal{J}} \max\{\mathrm{OPT}(i-1, J'), \sum_{j \in J' \setminus J} p_{ij}\}$. Using this recurrence relation a simple dynamic program can be formulated that computes the values $\mathrm{OPT}(i, J)$. It holds that $\mathrm{OPT}(I) = \mathrm{OPT}(m, \emptyset)$ and as usual for dynamic programs an optimal schedule can be recovered via backtracking. The running time of such a program can be bounded by $2^{\mathcal{O}(n)} \times \mathcal{O}(m)$, yielding the following trivial result:

Remark 5. $R||C_{\max}$ is FPT for the parameter n.

The second dynamic program is based on iterating through the jobs. Let $\lambda \in \mathbb{Z}_{\geq 0}^{\mathcal{M}}$. We call λ a *load vector* and say that a schedule σ fulfils λ, if $\lambda_i = \sum_{j \in \sigma^{-1}(i)} p_{ij}$. For $j \in [n]$ let $\Lambda(j)$ be the set of load vectors that are fulfilled by some schedule for the subinstance $I[[j], \mathcal{M}]$, assuming $\mathcal{J} = [n]$. Then $\Lambda(j)$ can also be defined recursively as the set of vectors λ with $\lambda_{i^*} = \lambda'_{i^*} + p_{i^* j}$ and $\lambda_i = \lambda'_i$ for $i \neq i^*$, where $i^* \in M(j)$ and $\lambda' \in \Lambda(j-1)$. Using this, a simple dynamic program can be formulated that computes $\Lambda(j)$ for all $j \in [n]$. $\mathrm{OPT}(I)$ can be recovered from $\Lambda(n)$ and a corresponding schedule can be found via backtracking. Let there be a bound L for the number of distinct loads that can occur on each machine, i.e., $|\{\sum_{j \in \sigma^{-1}(i)} p_{ij} \mid \sigma \text{ schedule for } I\}| \leq L$ for each $i \in \mathcal{M}$. Then the running time can be bounded by $L^{\mathcal{O}(m)} \times \mathcal{O}(n)$, yielding:

Remark 6. $R||C_{\max}$ is FPT for the pair of parameters m and k with $k \in \{\mathrm{OPT}, \max_i |J(i)|\}$.

For this note that both OPT and $2^{\max_i |J(i)|}$ are bounds for the number of distinct loads that can occur on any machine. This dynamic program can also be used to get a simple FPTAS for $R||C_{\max}$ for the case when the number of machines m is constant. For this let B be an upper bound of $\mathrm{OPT}(I)$ with $B \leq 2\mathrm{OPT}$. Such a bound can be found with the 2-approximation by Lenstra et al. [15]. Moreover let $\varepsilon > 0$ and $\delta = \varepsilon/2$. By rounding the processing time of every job up to the next integer multiple of $\delta B/n$ we get an instance I' whose optimum makespan is at most $\varepsilon\mathrm{OPT}(I)$ bigger than $\mathrm{OPT}(I)$. The dynamic program can easily be modified to only consider load vectors for I', where all loads are bounded by $(1 + \delta/n)B$. Therefore there can be at most $n/\delta + 2$ distinct load values for any machine and an optimal schedule for I' can be found in time $(n/\varepsilon)^{\mathcal{O}(m)} \times \mathcal{O}(n)$. The schedule can trivially be transformed into a schedule for the original instance without an increase in the makespan.

Tree decompostion and treewidth. A *tree decomposition* of a graph G is a pair $(T, \{X_t \mid t \in V(T)\})$, where T is a tree, $X_t \subseteq V(G)$ for each $t \in V(t)$ is a set of vertices of G, called a bag, and the following three conditions hold:

(i) $\bigcup_{t \in V(T)} X_t = V(G)$
(ii) $\forall \{u, v\} \in E(G) \exists t \in V(T) : u, v \in X_t$
(iii) For every $u \in V(G)$ the set $T_u := \{t \in V(T) \mid u \in X_t\}$ induces a connected subtree of T.

The *width* of the decomposition is $\max_{t \in V(T)}(|X_t| - 1)$, and the *treewidth* $\mathrm{tw}(G)$ of G is the minimum width of all tree decompositions of G. It is well known that forests are exactly the graphs with treewidth one, and that the treewidth of G is at least as big as the biggest clique in G minus 1. More precisely, for each set of vertices $V' \subseteq V(G)$ inducing a clique in G, there is a node $t \in V(T)$ with $V' \subseteq X_t$ (see e.g. [4]). For a given graph and a value k it can be decided in FPT time (and linear in $|V(G)|$) whether the treewidth of G is at most k and in the affirmative case a corresponding tree decomposition with $\mathcal{O}(k|V(G)|)$ nodes can be computed [3]. However, deciding whether a graph has a treewidth of at most k, is NP-hard [1].

Branch decomposition and rankwidth. It is easy to see that graphs with a small treewidth are sparse. Probably the most studied parameter for *dense* graphs is the cliquewidth $\mathrm{cw}(G)$. In this paper however we are going to consider a related parameter called the rankwidth $\mathrm{rw}(G)$. These two parameters are equivalent in the sense that $\mathrm{rw}(G) \leq \mathrm{cw}(G) \leq 2^{\mathrm{rw}(G)+1} - 1$ [19]. Furthermore it is known that $\mathrm{cw}(G) \leq 3 \times 2^{\mathrm{tw}(G)-1}$ [5]. On the other hand $\mathrm{tw}(G)$ cannot be bounded by any function in $\mathrm{cw}(G)$ or $\mathrm{rw}(G)$, which can easily be seen by considering complete graphs.

A *cut* of G is a partition of $V(G)$ into two subsets. For $X, Y \subseteq V(G)$ let $A_G[X, Y] = (a_{xy})$ be the $|X| \times |Y|$ adjacency submatrix induced by X and Y, i.e., $a_{xy} = 1$ if $\{x, y\} \in E(G)$ and $a_{xy} = 0$ otherwise for $x \in X$ and $y \in Y$. The *cut rank* of (X, Y) is the rank of $A_G[X, Y]$ over the field with two elements GF(2) and denoted by $\mathrm{cutrk}_G(X, Y)$. A *branch decomposition* of $V(G)$ is a pair (T, η), where T is a tree with $|V(G)|$ leaves whose internal nodes have all degree 3, and η is a bijection from $V(G)$ to the leafs of T. For each $e = \{s, t\} \in E(T)$ there is an induced cut $\{X_s, X_t\}$ of G: For $x \in \{s, t\}$ the set X_x contains exactly the nodes $\eta^{-1}(\ell)$, where $\ell \in V(T)$ is a leaf that is in the same connected component of T as x, if e is removed. Now the *width* of e (with respect to cutrk_G) is $\mathrm{cutrk}_G(X_s, X_t)$ and the *rankwidth* of the decomposition (T, η) is the maximum width over all edges of T. The *rankwidth* of G is the minimum rankwidth of all branch decompositions of G. It is well known that the cliquewidth of a complete graph is equal to 1 and this is also true for the rankwidth. For a given graph and fixed k there is an algorithm that finds a branch decomposition of width k in FPT-time (cubic in $|V(G)|$), or reports correctly that none exists [10].

3 Treewidth Results

We start with some basic relationships between different restriction parameters for $R||C_{\max}$, especially the treewidths of the different graphs for a given instance. Similar relationships have been determined for the three graphs in the context of constraint satisfaction.

Remark 7. $\mathrm{tw_p} \geq \max_i |J(i)|$ and $\mathrm{tw_d} \geq \max_j |M(j)|$.

To see this note that the sets $J(i)$ and $M(j)$ are cliques in the primal and dual graphs, respectively.

Remark 8. $\mathrm{tw_i} \leq \mathrm{tw_p}+1$ and $\mathrm{tw_i} \leq \mathrm{tw_d}+1$. On the other hand $\mathrm{tw_p} \leq (\mathrm{tw_i}+1) \max_i |J(i)|-1$ and $\mathrm{tw_d} \leq (\mathrm{tw_i}+1) \max_j |M(j)| - 1$.

These properties were pointed out by Kalaitis and Vardi [13] in a different context. Note that this Remark together with Theorem 1 implies the results of Theorem 2 concerning the parameter $\max_i |J(i)|$. Furthermore, in the case of $P||C_{\max}$ with only 1 job and m machines, or n jobs and only 1 machine the primal graph has treewidth 0 or $n-1$ and the dual $m-1$ or 0, respectively, while the incidence graph in both cases has treewidth 1.

Dynamic Programs

We show how a tree decomposition $(T, \{X_t \mid t \in V(T)\})$ of width k for any one of the three graphs can be used to design a dynamic program for the corresponding instance I of $R||C_{\max}$. Selecting a node as the root of the decompostion, the dynamic program works in a bottom-up manner from the leaves to the root. We assume that the decomposition has the following simple form: For each leaf node $t \in V(T)$ the bag X_t is empty and we fix one of these nodes as the root a of T. Furthermore each internal node t has exactly two children $\ell(t)$ and $r(t)$ (left and right), and each node $t \neq a$ has one parent $p(t)$. We denote the descendants of t with $\mathrm{desc}(t)$. A decomposition of this form can be generated from any other one without increasing the width and growing only linearly in size through the introduction of dummy nodes. The bag of a dummy node is either empty or identical to the one of its parent.

For each of the graphs and each node $t \in V(T)$ we define sets $\check{J}_t \subseteq \mathcal{J}$ and $\check{M}_t \subseteq \mathcal{M}$ of *inactive* jobs and machines along with sets J_t and M_t of *active* jobs and machines. The active jobs and machines in each case are defined based on the respective bag X_t, and the inactive ones have the property that they were active for a descendant $t \in \mathrm{desc}(t)$ of t but are not at t. In addition there are *nearly inactive* jobs $\tilde{J}_t$ and machines $\tilde{M}_t$, which are the jobs and machines that are deactivated when going from t to its parent $p(t)$ (for $t = a$ we assume them to be empty). The sets are defined so that certain conditions hold. The first two are that the (nearly) inactive jobs may only be processed on active or inactive machines, and the (nearly) inactive machines can only process active or inactive jobs:

$$M(\check{J}_t \cup \tilde{J}_t) \subseteq M_t \cup \check{M}_t \tag{1}$$

$$J(\check{M}_t \cup \tilde{M}_t) \subseteq J_t \cup \check{J}_t \tag{2}$$

Where $M(J^*) = \bigcup_{j \in J^*} M(j)$ and $J(M^*) = \bigcup_{i \in M^*} J(i)$ for any sets $J^* \subseteq \mathcal{J}$ and $M^* \subseteq \mathcal{M}$. Furthermore the (nearly) inactive jobs and machines of the children of an internal t form a disjoint union of the inactive jobs and machines of t, respectively:

$$\check{J}_t = \check{J}_{\ell(t)} \mathbin{\dot\cup} \tilde{J}_{\ell(t)} \mathbin{\dot\cup} \check{J}_{r(t)} \mathbin{\dot\cup} \tilde{J}_{r(t)} \tag{3}$$

$$\check{M}_t = \check{M}_{\ell(t)} \mathbin{\dot\cup} \tilde{M}_{\ell(t)} \mathbin{\dot\cup} \check{M}_{r(t)} \mathbin{\dot\cup} \tilde{M}_{r(t)} \tag{4}$$

Where $A \mathbin{\dot\cup} B$ for any two sets A, B emphasizes that the union $A \cup B$ is disjoint, i.e., $A \cap B = \emptyset$. Now at each node of the decomposition the basic idea is to perform three steps:

(i) Combine the information from the children (for internal nodes).
(ii) Consider the nearly inactive jobs and machines:

- *Primal and incidence graph:* Try all possible ways of scheduling active jobs on nearly inactive machines.
- *Dual and incidence graph:* Try all possible ways of scheduling nearly inactive jobs on active machines.

(iii) Combine the information from the last two steps.

For the second step the dynamic programs described in Section 2 are used as subprocedures. We now consider each of the three graphs.

The primal graph. In the primal graph all the vertices are jobs, and we define the active jobs of a tree node t to be exactly the jobs that are included in the respective bag, i.e., $J_t = X_t$. The inactive jobs are those that are not included in X_t but are in a bag of some descendant of t and the nearly inactive one are those that are active at t but inactive at $p(t)$, i.e., $\check{J}_t = \{j \in \mathcal{J} \mid j \notin X_t \wedge \exists t' \in \mathrm{desc}(t) : j \in X_{t'}\}$ and $\tilde{J}_t = J_t \setminus J_{p(t)}$. Moreover the inactive machines are the ones on which some inactive job may be processed, and the (nearly in-)active machines are those that can process (nearly in-)active jobs and are not inactive, i.e., $\check{M}_t = M(\check{J}_t)$, $M_t = M(J_t) \setminus \check{M}_t$ and $\tilde{M}_t = M(\tilde{J}_t) \setminus \check{M}_t$. For these definitions we get:

Lemma 9. *The conditions (1)-(4) hold, as well as:*

$$J(\tilde{M}_t) \subseteq J_t \tag{5}$$

$$M(\check{J}_t \cup \tilde{J}_t) = \tilde{M}_t \cup \check{M}_t \tag{6}$$

Proof. (1) and (6):

$$M(\check{J}_t \cup \tilde{J}_t) = M(\check{J}_t) \cup M(\tilde{J}_t) = \check{M}_t \cup (M(\tilde{J}_t) \setminus \check{M}_t) = \check{M}_t \cup \tilde{M}_t$$

This yields (6) and (6) implies (1).

(2) and (5): Let $i \in \check{M}_t \cup \tilde{M}_t$ and $j \in J(i)$. We first consider the case that $i \in \check{M}_t$. Then there is a job $j' \in \check{J}_t$ with $i \in M(j')$. If $j = j'$, we have $j \in \check{J}_t$ and otherwise $\{j, j'\} \in E(G)$. Because of (T2) there is a node $t' \in V(T)$ with $j, j' \in J_{t'}$. Since $j' \in \check{J}_t$, we have $j' \notin J_t$. This together with (T3) gives $t' \in \mathrm{desc}(t)$. Now $j \notin J_t$ implies $j \in \check{J}_t$. Therefore we have $j \in J_t \cup \check{J}_t$. Next we consider he case that $i \in \tilde{M}_t$. In this case there is a job $j' \in \tilde{J}_t$ with $i \in M(j')$ and for each job $j'' \in \check{J}_t$ we have $i \notin M(j'')$. If $j = j'$ we have $j \in J_t$ and otherwise $\{j, j'\} \in E(G)$. Because of (T2) there is again a node $t' \in V(T)$ with $j, j' \in J_{t'}$. Since $j, j' \notin \check{J}_t$, $j' \notin J_{p(t)}$ and $j' \in J_t$ we get $j \in J_t$ using (T3). This also implies (5).

(3): All but $(\check{J}_{\ell(t)} \cup \tilde{J}_{\ell(t)}) \cap (\check{J}_{r(t)} \cup \tilde{J}_{r(t)}) = \emptyset$ follows directly from the definitions. Assuming there is a job $j \in (\check{J}_{\ell(t)} \cup \tilde{J}_{\ell(t)}) \cap (\check{J}_{r(t)} \cup \tilde{J}_{r(t)})$ we get $j \in J_t$ because of (T3), yielding a contradiction.

(4): Because of (3) and the definitions we get $\check{M}_t = \check{M}_{\ell(t)} \cup \tilde{M}_{\ell(t)} \cup \check{M}_{r(t)} \cup \tilde{M}_{r(t)}$, and $\check{M}_{s(t)} \cap \tilde{M}_{s(t)}$ for $s \in \{\ell, r\}$ is clear by definition. Therefore it remains to show $(\check{M}_{\ell(t)} \cup \tilde{M}_{\ell(t)}) \cap (\check{M}_{r(t)} \cup \tilde{M}_{r(t)}) = \emptyset$. We assume that there is a machine i in this cut. Then there are jobs $j_s \in \check{J}_{s(t)} \cup \tilde{J}_{s(t)}$ for $s \in \{\ell, r\}$ with $i \in M(j_s)$. We have $\{j_\ell, j_r\} \in E$ and

because of (T2) there is a node t' with $j_\ell, j_r \in J_{t'}$. Because of $j_\ell, j_r \notin J_t$ and (T3) we have a contradiction. $\square$

For $J \subseteq \mathcal{J}$ and $M \subseteq \mathcal{M}$ let $\Gamma(J, M) = \{J' \subseteq J \,|\, \forall j \in J' : M(j) \cap M \neq \emptyset\}$. Let $t \in V(T)$, $J \in \Gamma(J_t, \check{M}_t)$ and $J' \in \Gamma(J_t \setminus \tilde{J}_t, \check{M}_t \cup \tilde{M}_t)$. We set $S(t, J)$ and $\tilde{S}(t, J')$ to be the sets of feasible schedules for the instances $I[\check{J}_t \cup J, \check{M}_t]$ and $I[\check{J}_t \cup \tilde{J}_t \cup J', \check{M}_t \cup \tilde{M}_t]$ respectively. We will consider $\mathrm{OPT}(S(t, J))$ and $\mathrm{OPT}(\tilde{S}(t, J'))$.

First note that $\mathrm{OPT}(I) = \mathrm{OPT}(S(a, \emptyset))$, where a is the root of T. Moreover, for a leaf node t there are neither jobs nor machines and $\mathrm{OPT}(S(t, \emptyset)) = \mathrm{OPT}(\tilde{S}(t, \emptyset)) = \mathrm{OPT}(\{\emptyset\}) = 0$ holds. Hence let t be a non-leaf node. We first consider how $\mathrm{OPT}(S(t, J))$ can be computed from the children of t (Step (i)). Due to Property (iii) of the tree decomposition and (1) the jobs from J are already active on at least one of the direct descendants of t. Because of this and (4), J may be split in two parts $J_\ell \dot\cup J_r = J$, where $J_s \in \Gamma(J_{s(t)} \setminus \tilde{J}_{s(t)}, \check{M}_{s(t)} \cup \tilde{M}_{s(t)})$ for $s \in \{\ell, r\}$. Let $\Phi(J)$ be the set of such pairs (J_ℓ, J_r). From (3), (4) and (6) we get:

Lemma 10. $\mathrm{OPT}(S(t, J)) = \min_{(J_\ell, J_r) \in \Phi(J)} \max_{s \in \{\ell, r\}} \mathrm{OPT}(\tilde{S}(s(t), J_s))$.

Proof. Let $\sigma^* \in S(t, J)$ be optimal. Since $J \subseteq J_t$, we have $J \cap \tilde{J}_{s(t)} = \emptyset$ for $s \in \{\ell, r\}$. Let $J_s^* = \sigma'^{*-1}(\check{M}_{s(t)} \cup \tilde{M}_{s(t)}) \cap J$. Because of (4) we have $J = J_\ell^* \dot\cup J_r^*$ and $J_s^* \in \Gamma(J_{s(t)} \setminus \tilde{J}_{s(t)}, \check{M}_{s(t)} \cup \tilde{M}_{s(t)})$ obviously holds. Let $\sigma_s^* = \sigma^*|_{J_s' \cup \tilde{J}_{s(t)}}$. Because of (6) we have $\sigma_s^* \in \tilde{S}(s(t), J_s^*)$ and (3) implies $\sigma^* = \sigma_\ell^* \cup \sigma_r^*$. This yields:

$$\begin{aligned}
\mathrm{OPT}(S(t, J)) &= C_{\max}(\sigma^*) \\
&= \max_{s \in \{\ell, r\}} C_{\max}(\sigma_s^*) \\
&\geq \max_{s \in \{\ell, r\}} \mathrm{OPT}(\tilde{S}(s(t), J_s^*)) \\
&\geq \min_{(J_\ell, J_r) \in \Phi(J)} \max_{s \in \{\ell, r\}} \mathrm{OPT}(\tilde{S}(s(t), J_s))
\end{aligned}$$

Now let $(J_\ell, J_r) \in \Phi(J)$ minimizing the righthand side of the equation and $\sigma_s \in \tilde{S}(s(t), J_s)$ optimal. Then (3) and (4) imply that $\sigma := \sigma_\ell \cup \sigma_r$ is in $S(t, J)$. Therefore we have $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. Since $C_{\max}(\sigma)$ also equals the right hand side of the equation the claim follows. $\square$

Consider the computation of $\mathrm{OPT}(\tilde{S}(t, J'))$ (Step (iii)). We may split J' and $\tilde{J}_t$ into a set going to the nearly inactive and a set going to the inactive machines. We set $\Psi(J')$ to be the set of pairs (A, X) with $J' \cup \tilde{J}_t = A \dot\cup X$, $A \in \Gamma(\tilde{J}_t \cup J', \tilde{M}_t)$ and $X \in \Gamma(\tilde{J}_t \cup J', \check{M}_t)$. Because of (3)-(5) we have:

Lemma 11. $\mathrm{OPT}(\tilde{S}(t, J')) = \min_{(A, X) \in \Psi(J')} \max\{\mathrm{OPT}(S(t, X)), \mathrm{OPT}(I[A, \tilde{M}_t])\}$.

Proof. Let $\sigma^* \in \tilde{S}(t, J')$ be optimal. Because of (5) we have $\sigma^{*-1}(\tilde{M}_t) \subseteq J' \cup \tilde{J}_t$. We set $A^* = \sigma^{*-1}(\tilde{M}_t)$ and $X^* = (J' \cup \tilde{J}_t) \setminus A$. Then $(A^*, X^*) \in \Psi(J')$. Let $\check{\sigma}^* = \sigma^*|_{\check{J}_t \cup X^*}$ and $\tilde{\sigma}^* = \sigma^*|_A$. Then $\check{\sigma}^* \in S(t, X^*)$ and $\tilde{\sigma}^*$ is a feasible schedule for $I[A^*, \tilde{M}_t]$. Because of (3)

and (4), we have $\sigma = \check{\sigma}^* \dot{\cup} \tilde{\sigma}^*$ and:

$$\begin{aligned}\mathrm{OPT}(\tilde{S}(t, J')) &= C_{\max}(\sigma^*) \\ &= \max\{C_{\max}(\check{\sigma}^*), C_{\max}(\tilde{\sigma}^*)\} \\ &\geq \max\{\mathrm{OPT}(S(t, X^*)), \mathrm{OPT}(I[A^*, \tilde{M}_t])\} \\ &\geq \min_{(A,X)\in\Psi(J')} \max\{\mathrm{OPT}(S(t, X)), \mathrm{OPT}(I[A, \tilde{M}_t])\}\end{aligned}$$

Now let $(A, X) \in \Psi(J')$ minimizing the right hand side of the equation, $\check{\sigma} \in S(t, X)$ and $\tilde{\sigma}$ a feasible schedule for $I[A, \tilde{M}_t]$. Then (3) and (4) yield $\sigma := \check{\sigma} \dot{\cup} \tilde{\sigma} \in \tilde{S}(t, J')$, and therefore $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. Since $C_{\max}(\sigma)$ also equals the right hand side of the equation, the claim follows. □

Determining the values $\mathrm{OPT}(I[A, \tilde{M}_t])$ corresponds to Step (ii). Note that these values can be computed using the first dynamic program from Section 2 in time $2^{\mathcal{O}(k)} \times \mathcal{O}(m)$.

The dual graph. For the dual graph the (in-)active jobs and machines are defined dually: The active machines for a tree node t are the ones in the respective bag, the inactive machines are those that were active for some descendant but are not active for t, and the nearly inactive machines are those that are active at t but inactive at its parent, i.e., $M_t = X_t$, $\check{M}_t = \{i \in \mathcal{M} \mid i \notin M_t \wedge \exists t' \in \mathrm{desc}(t) : i \in X_{t'}\}$ and $\tilde{M}_t = M_t \setminus \check{M}_{p(t)}$. Furthermore the inactive jobs are those that may be processed on some inactive machine and the (nearly in-)active ones are those that can be processed on some (nearly in-)active machine and are not inactive, i.e., $\check{J}_t = J(\check{M}_t)$, $J_t = J(M_t) \setminus \check{J}_t$ and $\tilde{J}_t = J(\tilde{M}_t) \setminus \check{J}_t$. With these definitions we get analogously to Lemma 9:

Lemma 12. *The conditions (1)-(4) hold, as well as:*

$$M(\tilde{J}_t) \subseteq M_t \tag{7}$$

$$J(\check{M}_t \cup \tilde{M}_t) = \tilde{J}_t \cup \check{J}_t \tag{8}$$

□

We will need some extra notation. Like we did in Section 2 we will consider load vectors $\lambda \in \mathbb{Z}^M_{\geq 0}$, where $M \subseteq \mathcal{M}$ is a set of machines. We say that a schedule σ fulfils λ, if $\lambda_i = \sum_{j\in\sigma^{-1}(i)} p_{ij}$ for each $i \in M$. For any set S of schedules for I we denote the set of load vectors for M that are fulfilled by at least one schedule from S with $\Lambda(S, M)$. Furthermore we denote the set of all schedules for I with $S(I)$, and for a subset of jobs $J \subseteq \mathcal{J}$, we write $\Lambda(J, M)$ as a shortcut for $\Lambda(S(I[J, M]), M)$. Let $t \in V(T)$. We set $S(t) = S(I[\check{J}_t, \check{M}_t \cup M_t])$ and $\tilde{S}(t) = S(I[\check{J}_t \cup \tilde{J}_t, \check{M}_t \cup M_t])$. Moreover, for $\lambda \in \Lambda(S(t), M_t)$ and $\lambda' \in \Lambda(\tilde{S}(t), M_t)$ we set $S(t, \lambda) \subseteq S(t)$ and $\tilde{S}(t, \lambda') \subseteq \tilde{S}(t)$ to be those schedules that fulfil λ and λ' respectively. We now consider $\mathrm{OPT}(S(t, \lambda))$ and $\mathrm{OPT}(\tilde{S}(t, \lambda'))$.

First note $\mathrm{OPT}(I) = \mathrm{OPT}(S(a, \emptyset))$. Moreover, for a leaf node t we have neither jobs nor machines and $\Lambda(S(t), M_t) = \Lambda(\tilde{S}(t), M_t) = \{\emptyset\}$. Therefore $\mathrm{OPT}(S(t, \emptyset)) = \mathrm{OPT}(\tilde{S}(t, \emptyset)) = \mathrm{OPT}(\{\emptyset\}) = 0$. Hence let t be a non-leaf node. Again, we first consider how $\mathrm{OPT}(S(t, \lambda))$ can be computed from the children of t. Because of (3) λ may be split into a left and a right part. For two machine sets M, M' let $\tau_{M,M'} : \mathbb{Z}^M_{\geq 0} \to \mathbb{Z}^{M'}_{\geq 0}$ be a trasformation

function for load vectors, where the i-th entry of $\tau_{M,M'}(\lambda)$ equals λ_i for $i \in M \cap M'$ and 0 otherwise. We set $\Xi(\lambda)$ to be the set of pairs $(\lambda_\ell, \lambda_r)$ with $\lambda = \tau_{M_{\ell(t)},M_t}(\lambda_\ell) + \tau_{M_{r(t)},M_t}(\lambda_r)$, and $\lambda_s \in \Lambda(\tilde{S}(s(t)), M_{s(t)})$ for $s \in \{\ell, r\}$. Because of (1), (3) and (4), we have:

Lemma 13. $\mathrm{OPT}(S(t,\lambda)) = \min_{(\lambda_\ell,\lambda_r)\in\Xi(\lambda)} \max_{s\in\{\ell,r\}} \mathrm{OPT}(\tilde{S}(s(t), \lambda_s))$.

Proof. Let $\sigma^* \in S(t,\lambda)$ be optimal. Because of (1) we have $\sigma^*(\check{J}_{s(t)} \cup \tilde{J}_{s(t)}) \subseteq M_{s(t)} \cup \check{M}_{s(t)}$ for $s \in \{\ell, r\}$. Let $\sigma^*_s = \sigma^*|_{\check{J}_{s(t)}\cup\tilde{J}_{s(t)}}$ and λ^*_s the load vector that σ^*_s fulfils on $M_{s(t)}$. Then we have $\sigma^*_s \in \tilde{S}(s(t), \lambda^*_s)$ and $(\lambda^*_\ell, \lambda^*_r) \in \Xi(\lambda)$. Because of (3) and (4) we have $\sigma^* = \sigma^*_\ell \mathbin{\dot\cup} \sigma^*_r$. Because of the objective function we have:

$$\begin{aligned}
\mathrm{OPT}(S(t,\lambda)) &= C_{\max}(\sigma^*) \\
&= \max_{s\in\{\ell,r\}} C_{\max}(\sigma^*_s) \\
&\geq \max_{s\in\{\ell,r\}} \mathrm{OPT}(\tilde{S}(s(t), \lambda^*_s)) \\
&\quad \min_{(\lambda_\ell,\lambda_r)\in\Xi(\lambda)} \max_{s\in\{\ell,r\}} \mathrm{OPT}(\tilde{S}(s(t), \lambda_s))
\end{aligned}$$

Now let $(\lambda_\ell, \lambda_r) \in \Xi(\lambda)$ minimizing the right hand side of the equation and $\sigma_s \in \tilde{S}(s(t), \lambda_s)$ optimal. Then $\sigma := \sigma_\ell \cup \sigma_r$ is in $S(t,\lambda)$ and $C_{\max}(\sigma)$ equals the right hand side of the equation. Since furthermore $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$ the claim follows. □

Now we consider $\mathrm{OPT}(\tilde{S}(t,\lambda'))$. We may split λ' into the load due to inactive and that due to nearly inactive jobs. Note that the nearly inactive jobs can only be processed by active machines (7). We set $\Upsilon(\lambda')$ to be the set of pairs (α, ξ) with $\lambda' = \alpha + \xi$, $\alpha \in \Lambda(\tilde{J}_t, M_t)$ and $\xi \in \Lambda(S(t), M_t)$. Now (3), (4) and (7) yield:

Lemma 14. $\mathrm{OPT}(\tilde{S}(t,\lambda')) = \min_{(\alpha,\xi)\in\Upsilon(\lambda')} \max(\{\mathrm{OPT}(S(t,\xi))\} \cup \{\lambda'_i \mid i \in M_t\})$.

Proof. Let $\sigma^* \in \tilde{S}(t,\lambda')$ be optimal. Then (7) implies $\sigma^*(\tilde{J}_t) \subseteq M_t$. We set $\tilde{\sigma}^* = \sigma^*|_{\tilde{J}_t}$ and $\check{\sigma}^* = \sigma^*|_{\check{J}_t}$. Furthermore let α^* be the load vector fulfilled by $\tilde{\sigma}^*$ and ξ^* the one fulfilled by $\check{\sigma}^*$ on M_t. Then $\tilde{\sigma}^*$ is a feasible schedule for $I[\tilde{J}_t, M_t]$ fulfilling α^*, $\check{\sigma}^* \in S(t,\xi^*)$ and $(\alpha^*, \xi^*) \in \Upsilon(\lambda')$. Furthermore (3) yields $\sigma^* = \tilde{\sigma}^* \mathbin{\dot\cup} \check{\sigma}^*$. We get:

$$\begin{aligned}
\mathrm{OPT}(\tilde{S}(t,\lambda')) &= C_{\max}(\sigma^*) \\
&= \max(\{C_{\max}(\check{\sigma}^*)\} \cup \{\lambda'_i \mid i \in M_t\}) \\
&\geq \max(\{\mathrm{OPT}(S(t,\xi^*))\} \cup \{\lambda'_i \mid i \in M_t\}) \\
&\geq \min_{(\alpha,\xi)\in\Upsilon(\lambda')} \max(\{\mathrm{OPT}(S(t,\xi))\} \cup \{\lambda'_i \mid i \in M_t\})
\end{aligned}$$

Now let $(\alpha, \xi) \in \Upsilon(\lambda')$ minimizing the right hand side of the equation, $\check{\sigma} \in S(t,\xi)$ and $\tilde{\sigma}$ a feasible schedule for $I[\tilde{J}_t, M_t]$ fulfilling α. Then $\sigma := \check{\sigma} \cup \tilde{\sigma} \in \tilde{S}(t,\lambda')$ and therefore $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. Since $C_{\max}(\sigma)$ also equals the right hand side of the equation, the claim follows. □

The set $\Lambda(\tilde{J}_t, M_t)$ can be computed using the second dynamic program described in Section 2 in time $L^{\mathcal{O}(k)} \times \mathcal{O}(n)$ if L is again a bound on the number of distinct loads that can occur on each machine.

The incidence graph. For the incidence graph we combine the ideas that we used for the two other graphs. The situation is slightly more complicated because we have to handle the jobs and machines simultaneously. All the job sets are defined like in the primal, and all the machine sets like in the dual graph case. With these definitions the conditions (1)-(4) follow almost directly from the definitions together with (T2) and (T3). The proofs for the recurrence relations in this paragraph have the same structure as the proofs for the other recurrence relations and no new ideas are needed. Therefore they are omitted.

Let $t \in V(t)$, $J \in \Gamma(J_t, \check{M}_t)$ and $J' \in \Gamma(J_t \setminus \tilde{J}_t, \check{M}_t \cup \tilde{M}_t)$. We set $S(t, J)$ to be the set of feasible schedules σ for $I[\check{J}_t \cup J, \check{M}_t \cup M_t]$ that schedule the jobs from J on inactive machines, i.e., $\sigma(j) \in \check{M}_t$ for each $j \in J$. Moreover, $\tilde{S}(t, J')$ is the set of schedules for $I[\check{J}_t \cup \tilde{J}_t \cup J', \check{M}_t \cup M_t]$ that schedule the jobs from J' on (nearly) inactive machines $\tilde{M}_t \cup \check{M}_t$. The sets of schedules that in addition fulfil a load vector $\lambda \in \Lambda(S(t, J), M_t)$ or $\lambda' \in \Lambda(\tilde{S}(t, J')$ are denoted by $S(t, J, \lambda)$ and $\tilde{S}(t, J', \lambda')$. We consider $\mathrm{OPT}(S(t, J, \lambda))$ and $\mathrm{OPT}(\tilde{S}(t, J', \lambda'))$.

First note $\mathrm{OPT}(I) = \mathrm{OPT}(S(a, \emptyset, \emptyset))$. For a leaf note t there are neither jobs nor machines and therefore $\mathrm{OPT}(S(t, \emptyset, \emptyset)) = \mathrm{OPT}(\emptyset)) = 0$. Hence let t be a non-leaf node. Like before, we first consider $\mathrm{OPT}(S(t, J, \lambda))$. Both J and λ may be split into a left and a right part and we set $\Phi(J)$ like before. Moreover, for $(J_\ell, J_r) \in \Phi(J)$ we define $\Xi(\lambda, (J_\ell, J_r))$ to be the set of pairs $(\lambda_\ell, \lambda_r)$ with $\lambda_s \in \Lambda(\tilde{S}(s(t), J_s), M_{s(t)})$ for $s \in \{\ell, r\}$. Due to (1)-(4) we have:

Lemma 15. $\mathrm{OPT}(S(t, J, \lambda)) = \min_{(J_\ell, J_r), (\lambda_\ell, \lambda_r)} \max_{s \in \{\ell, r\}} \mathrm{OPT}(\tilde{S}(s(t), J_s, \lambda_s))$. □

Next we consider $\mathrm{OPT}(\tilde{S}(t, J', \lambda'))$. The set J' again may be split into a part going to the inactive and a part going to the nearly inactive machines, while the nearly inactive jobs $\tilde{J}_t$ have to be split into a part going to the inactive and a part going to the active machines (note that in this case (7) does not hold). Therefore, we set $\Psi(J')$ to be the set of pairs (A, X) with $J' = A \dot{\cup} X$, $A \cap J' \in \Gamma(J', \tilde{M}_t)$, $A \cap \tilde{J}_t \in \Gamma(\tilde{J}_t, M_t)$ and $X \in \Gamma(\tilde{J}_t \cup J', \check{M}_t)$. The splitting of λ' is more complicated as well, because in this case all of the active machines may receive load from the nearly inactive jobs, and the nearly inactive machines may additionally receive load from the active but not nearly inactive jobs ((5) does not hold). Therefore we set $\Upsilon(\lambda', (A, X))$ to be the set of triplets (α, β, ξ) with $\alpha \in \Lambda(A \cap J', \tilde{M}_t)$, $\beta \in \Lambda(A \cap \tilde{J}_t, M_t)$, $\xi \in \Lambda(S(t, X), M_t)$ and $\lambda' = \tau_{\tilde{M}_t, M_t}(\alpha) + \beta + \xi$. Due to (3) and (4) we have:

Lemma 16. $\mathrm{OPT}(\tilde{S}(t, J', \lambda')) = \min_{(A,X), (\alpha, \beta, \xi)} \max(\{\mathrm{OPT}(S(t, X, \xi))\} \cup \{\lambda'(i) \mid i \in M_t\})$. □

Note that the sets $\Lambda(A \cap J', \tilde{M}_t)$ and $\Lambda(A \cap \tilde{J}_t, M_t)$ can be computed in time $L^{\mathcal{O}(k)}$ using the second dynamic program described in Section 2, if L is again a bound on the number of distinct loads that can occur on each machine.

Results. Using above arguments, we can design dynamic programs with running time $2^{\mathcal{O}(k)} \times \mathcal{O}(nm)$ in the primal case and $L^{\mathcal{O}(k)} \times \mathcal{O}(nm)$ in the dual and incidence graph cases. Optimal schedules can be found via backtracking proving the Theorems 1 and 2. Theorem 3 follows by the combination of the dynamic programs and a rounding scheme similar to that in Section 2.

4 Rankwidth Results

First we want to argue that there is not much to be gained when considering primal or dual graphs with bounded rankwidth. For this consider any instance I of $P|M(j)|C_{\max}$. By adding a job with processing time $\mathrm{OPT}(I)$ that can be processed on every machine, and a machine that can only process this new job, we get a modified instance I'. Any schedule for one of the instances can trivially be transformed into a schedule for the other without an increase in the makespan. However, while the rankwidth of the primal or dual graph of I could have been arbitrarily high, the rankwidth of the primal and dual graph of I' are both equal to one, because these graphs are complete.

We study the case when the rankwidth of the incidence graph is bounded by a constant k. Moreover we assume that also the number d of distinct job sizes is bounded by a constant, which we can do because of the following result. Let $\mathcal{I}$ be some class of instances of $P|M(j)|C_{\max}$, which is invariant with respect to changing the processing times of jobs and the introduction of copies of jobs.

Lemma 17 (Rounding Lemma)**.** *If there is a PTAS for instances from $\mathcal{I}$, for which the number of distinct processing times is bounded by a constant, then there is also a PTAS for any instance from $\mathcal{I}$.*

Proof. Let $I \in \mathcal{I}$, $\varepsilon > 0$ and B an upper bound of $\mathrm{OPT}(I)$ with $B \leq 2\mathrm{OPT}$. Such a bound B can be found in polynomial time for example with the 2-approximation by Lenstra et al. [15]. Moreover let $\delta := \min\{1/3, \varepsilon/7\}$. We call jobs j *big*, if $p_j > \delta B$ and otherwise *small*. Next, we construct a modified instance I'. This instance has the same machine set and for each big job j a job with the same restrictions and processing time $p'_j := \delta^2 B \lceil \frac{p_j}{\delta^2 B} \rceil$ is included in the job set. This yields $p'_j \leq p_j + \delta^2 B \leq (1+\delta)p_j$. For each small job j in I we introduce $\lceil \frac{np_j}{\delta B} \rceil \in \mathcal{O}(n)$ many jobs with the same restrictions as j, and with processing time $\frac{\delta B}{n}$.

Note that I' has a has at most $1/\delta + 1$ many distinct processing times and that $I' \in \mathcal{I}$. Moreover the size of I' is polynomial in the size of I.

Given an optimal solution of I, consider the solution of I' we get by scheduling both the big and the small jobs in I' the same way as there analogues in I. The big jobs on a machine can cause an increase of the processing time of at most factor $(1+\delta)$, while for each small job of I there may be an increase of at most $\frac{\delta B}{n}$. Therefore we get:

$$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) + \delta\mathrm{OPT}(I) + \delta B \leq (1+3\delta)\mathrm{OPT}(I)$$

Now given a PTAS for instances of $\mathcal{I}$ for which the number of distinct processing times is bounded by a constant, we can compute a schedule σ' for I' with $C_{\max}(\sigma) \leq (1+\delta)\mathrm{OPT}(I')$ in polynomial time. We use σ' to construct a schedule for σ for I. In this schedule the big jobs are assigned in the same way as there analogous in I'. For the small jobs we need some additional consideration. Let S and S' be the set of small jobs in I and I' respectively. Moreover, for $j \in S$ let $S'(j)$ be the set of small jobs that were inserted in I' due to j. The assignment of $S(j)$ in σ' can be seen as a *fractional* assignment of j. We find a rounding for this fractional assignment of the small jobs. For each machine i and small job $j \in S$ let $x_{ij} = |\{j' \in S(j) \mid \sigma'(j) = i\}|/|S(j)|$. Furthermore let t_i be the summed up processing time that machine i receives in the schedule σ' from small jobs, i.e., $t_i = |\{j' \in S' \mid \sigma'(j) = i\}|\frac{\delta B}{n}$.

Then (x_{ij}) is a solution of the following linear program:

$$\sum_{iM(j)} x_{ij} = 1 \qquad \forall j \in S \tag{9}$$

$$\sum_{j \in S} p_j x_{ij} \leq t_i \qquad \forall i \in \mathcal{M} \tag{10}$$

$$x_{ij} \geq 1 \qquad \forall j \in S, i \in \mathcal{M}$$

Using the rounding approach by Lenstra et al. [15], we can transform this in polynomial time into an integral solution fulfilling the constraint (9) and instead of (10) the modified constraint:

$$\sum_{j \in S} p_j x_{ij} \leq t_i + \max_{j \in S} p_j \qquad \forall i \in \mathcal{M}$$

We set σ to assign the small jobs according to (x_{ij}). Since $\max_{j \in S} p_j \leq \delta B$ we get $C_{\max}(\sigma) \leq C_{\max}(\sigma') + \delta B$ and together with the above considerations:

$$C_{\max}(\sigma) \leq ((1+\delta)(1+3\delta) + 2\delta)\mathrm{OPT}(I) \leq (1+\varepsilon)\mathrm{OPT}(I)$$

□

While all of the used techniques are well known they have—to the best of our knowledge—not been used in the indicated way up to now.

It can be easily seen that the class of instances of $P|M(j)|C_{\max}$, for which the rankwidth of the incidence graph is bounded by a constant k, is invariant with respect to changing the processing time of jobs and the introduction of copies of jobs.

Dynamic Program

We present a dynamic program to solve $P|M(j)|C_{\max}$ using a branch decomposition (T, η) with rankwidth k for the incidence graph. First we give some intuition on why a bounded rankwidth is useful.

For any Graph (V, E) and $X \subseteq V$, we say that $u, v \in V$ have the same *connection type with respect to* X if $N(u) \cap X = N(v) \cap X$. If X is clear from the context we say that u and v have the same connection type. Now, let $e = \{a, b\} \in E(T)$ be some edge of the branch decomposition and $\{X_{e,a}, X_{e,b}\}$ the respective cut of T, i.e., $X_{e,x}$ for $x \in \{a, b\}$ is the set of vertices of T that are in the same connected component as x when the edge e is removed. Then $\{X_{e,a}, X_{e,b}\}$ induces a partition of both the jobs and machines by $J_{e,x} := \{j \in \mathcal{J} \mid \eta(j) \in X_{e,x}\}$ and $M_{e,x} := \{i \in \mathcal{M} \mid \eta(j) \in X_{e,x}\}$ for $x \in \{a, b\}$.

Remark 18. Let $x, y \in \{a, b\}$ with $x \neq y$. The number of distinct connection types of $J_{e,x}$ with respect to $M_{e,y}$ is bounded by 2^k.

We actually use that the number of distinct connection types of the jobs is bounded.

In the rest of this section we first show how the branch decomposition can be used in a straightforward way to solve $P|M(j)|C_{\max}$ (with exponential running time). The basic idea for this is that each edge of the decomposition corresponds to a partition of the job and machine sets and an optimal solution may be found by trying all possible ways of

moving jobs between partitions. At the machine-leafs all arriving jobs have to be processed, with no jobs going out, and at the job-leafs all jobs have to be send away, with no jobs coming in. From this the procedure can work up to some root edge. Next we argue that it is sufficient to consider only certain locally defined classes of job sets. The crucial part here is that the number of these classes can be polynomially bounded, because the number of distinct sizes and connection types of jobs are constant.

Job sets. Let $e = \{a, b\} \in E(T)$ again be some edge of the tree T and $\{X_{e,a}, X_{e,b}\}$ the corresponding cut of T. We fix a schedule σ and make some basic observations. There is a set of jobs $\vec{J}_{a,b} \subseteq J_{e,a}$ that σ assigns to machines from $M_{e,b}$. We will use the intuition that $\vec{J}_{a,b}$ is sent through e from a to b (see also Figure 2). The node b may be an inner node or a leaf. Moreover, if b is a leaf, $\eta^{-1}(t)$ may be a job j^* or a machine i^*. In the first case σ sends no jobs to t, and j^* to a. In the second case no jobs are sent to a and the jobs send to b should be feasible on i^*. Now any set that is sent through an edge and arrives at an internal node, will be split into two parts: one going forth through the second and one through the third edge. And looking at it the other way around: Any set that is sent by a schedule through an edge coming from an inner node, is put together from two parts, one coming from the second and one coming from the third edge.

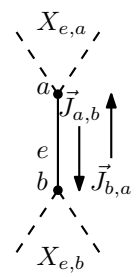

Figure 2: Edge e

We formalize this notion. Let t be an internal node of T, with neighbours $u, v, w \in V(T)$. Then for each pair of neighbours x, y of t there are job sets $\vec{J}_{x,t,y} \subseteq \vec{J}_{x,t} \cap \vec{J}_{t,y}$, such that:

$$\vec{J}_{u,t} = \vec{J}_{u,t,v} \dot\cup \vec{J}_{u,t,w} \qquad\qquad \vec{J}_{t,u} = \vec{J}_{v,t,u} \dot\cup \vec{J}_{w,t,u} \tag{11}$$

See also Figure 3. It is rather easy to see that sets $\vec{J}_{s,t}$ that are feasible at the leafs and fulfil the conditions (11) uniquely define a feasible schedule.

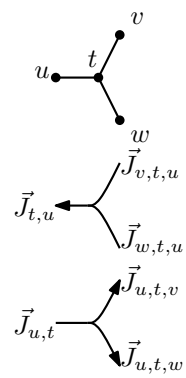

Figure 3: Job-sets are split.

Using these observations, we now discuss how (the value of) an optimal schedule can be found by considering different job sets that may be sent through the edges. For this we use an intuition of up and down, with a above and b below. Let $\check{J} = \vec{J}_{a,b} \subseteq J_{e,a}$ and $\hat{J} = \vec{J}_{b,a} \subseteq J_{e,b}$ be some candidate sets to be sent up and down respectively through e. We set $I_{e,x}(\hat{J}, \check{J}) = I[(J_{e,x} \setminus \vec{J}_{x,y}) \cup \vec{J}_{y,x}, M_{e,x}]$ for $x, y \in \{a, b\}$ with $x \neq y$, i.e., the subinstances of I induced by e, if $\hat{J}$ and $\check{J}$ are send up or down respectively. Note that the instance I is split into the two subinstances. Moreover let Γ_e be the set of pairs $(\hat{J}, \check{J})$. Then:

$$\mathrm{OPT}(I) = \min_{(\hat{J},\check{J})\in\Gamma_e} \max\{\mathrm{OPT}(I_{e,a}(\hat{J}, \check{J})), \mathrm{OPT}(I_{e,b}(\hat{J}, \check{J}))\} \tag{12}$$

We now consider the computation of $\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J}))$ for the two cases when b is an internal node or a leaf. If b is a leaf, it may correspond to a job or a machine, i.e., $\eta^{-1}(b) = j^*$ or $\eta^{-1}(b) = i^*$. In the first case we get $\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J})) = \infty$ if $\hat{J} \neq \{j^*\} = J_{e,b}$ or $\check{J} \neq \emptyset$, and $\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J})) = 0$ otherwise. In the second case $\hat{J}$ is empty since there are no jobs at b. We get $\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J})) = \sum_{j\in\check{J}} p_j$ if $\check{J} \subseteq M(i^*)$ and $\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J})) = \infty$ otherwise.

Now let b be an internal node that is connected to two lower nodes ℓ and r via edges e_ℓ and e_r (Figure 4). We say that ℓ and e_ℓ are on the left, while r and e_r are on the right. Recursively we assume that for any $(\hat{L}, \check{L}) \in \Gamma_{e_\ell}$ and $(\hat{R}, \check{R}) \in \Gamma_{e_r}$ we know $\mathrm{OPT}(I_{e_\ell,\ell}(\hat{L}, \check{L}))$ and $\mathrm{OPT}(I_{e_r,r}(\hat{R}, \check{R}))$ respectively. We want to identify the set $\Lambda_e(\hat{J}, \check{J})$ of tuples $(\hat{L}, \check{L}, \hat{R}, \check{R})$ that for fixed $(\hat{J}, \check{J})$ may occur in a schedule, i.e., fulfil condition (11) for all edges from $\{e, e_\ell, e_r\}$. For $\hat{J}$ it is clear which part is coming from the left and which from the right and we set $\hat{J}_\ell \subseteq J_{e_\ell,\ell}$ and $\hat{J}_r \subseteq J_{e_r,r}$ accordingly, such that $\hat{J} = \hat{J}_\ell \dot{\cup} \hat{J}_r$. The other four sets in which the job sets going up and down could be split can all be tried. More precisely, for each $\check{J}_\ell, \check{J}_r \subseteq \check{J}$ with $\check{J} = \check{J}_\ell \dot{\cup} \check{J}_r$, $\hat{L}_r \subseteq (J_{e_\ell,\ell} \cap \mathcal{J}) \setminus \hat{J}_\ell$ and $\hat{R}_\ell \subseteq (J_{e_r,r} \cap \mathcal{J}) \setminus \hat{J}_r$ the tuple $(\hat{L}_r \dot{\cup} \hat{J}_\ell, \check{J}_\ell \dot{\cup} \hat{R}_\ell, \hat{R}_\ell \dot{\cup} \hat{J}_r, \check{J}_r \dot{\cup} \hat{L}_r)$ is in $\Lambda_e(\hat{J}, \check{J})$ and the set is defined by such tuples. We get:

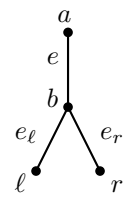

Figure 4: Inner node b

$$\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J})) = \min_{(\hat{L},\check{L},\hat{R},\check{R})} \max\{\mathrm{OPT}(I_{e_\ell,\ell}(\hat{L}, \check{L})), \mathrm{OPT}(I_{e_r,r}(\hat{R}, \check{R}))\} \tag{13}$$

Using these considerations $P|M(j)|C_{\max}$ can be solved by choosing a root edge $e^* = \{a^*, b^*\}$ that is incident to a leaf a^* corresponding to a job and designing a dynamic program working from leaf edges to the root edge using (13). Now (12) for $e = e^*$ together with the considerations for leaf nodes yield $\mathrm{OPT}(I) = \mathrm{OPT}(I_{e^*,b^*}(\emptyset, \eta^{-1}(a^*)))$. The running time of such an algorithm is exponential in the input length.

Classes of Jobs. Let $\check{J}, \check{J}' \subseteq J_{e,a}$. There are some cases in which $\check{J}$ and $\check{J}'$ are in some sense similar and it holds that $\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J})) = \mathrm{OPT}(I_{e,b}(\hat{J}, \check{J}'))$. This is the case if there is a bijection $\alpha : \check{J} \to \check{J}'$ such that j and $\alpha(j)$ have the same connection type with respect to $M_{e,b}$ and $p_j = p_{\alpha(j)}$ for each $j \in \hat{J}$. By this, an equivalence relation $\sim_{e,a}$ can be defined, and analogously a relation $\sim_{e,b}$. Now the observation (12) can be reformulated in terms of equivalence classes:

Lemma 19. $\mathrm{OPT}(I) = \min_{([\hat{J}],[\check{J}])} \max\{\min_{\hat{J}'} \mathrm{OPT}(I_{e,a}(\hat{J}', \check{J})), \min_{\check{J}'} \mathrm{OPT}(I_{e,b}(\hat{J}, \check{J}'))\}$. □

Note that in this equation equivalence classes $[\check{J}]$ and $[\hat{J}]$ are considered belonging to the relations $\sim_{e,a}$ and $\sim_{e,b}$ respectively. $\hat{J}$ and $\check{J}$ are arbitrary representatives of these classes. We will now develop a sensible representation for the equivalence classes.

We drop the notion of up and down for the following considerations, i.e., $b \in e$ is just one of two nodes of some edge e. We assume some ordering of the different processing times, with $p(i)$ denoting the i-th processing time for $i \in [d]$. Any set of jobs J' induces a vector $\lambda \in \mathbb{Z}^d_{\geq 0}$ where λ_i is the number of jobs in J' that have the i-th processing time, i.e., $\lambda_i = |\{j \in J' \,|\, p_j = p(i)\}|$. We set $p(\lambda) = \sum_{i\in[d]} p(i)\lambda_i$. Let $\kappa(e, b)$ be the number of connection types of jobs from $J_{e,b}$ with respect to $M_{e,a}$. Note that due to Remark 18 we get $\kappa(e, b) \leq 2^k$. Again assuming some ordering, for $i \in [\kappa(e, b)]$ let $\varphi_{e,b}(i)$ be the size vector induced by the i-th connection type of $J_{e,b}$ with respect to $M_{e,a}$ and $M_{e,a}(i) \subseteq M_{e,a}$ the machines from $M_{e,a}$ the respective jobs may be processed on. Moreover let $\varphi_{e,b} = (\varphi_{e,b}(1), \ldots, \varphi_{e,b}(\kappa(e, b)))$. Now the equivalence classes of $\sim_{e,b}$ can naturally be represented and characterized by vectors $\iota \leq \varphi_{e,b}$.

Remark 20. For each $e \in E(T)$ and $b \in e$ there are at most $n^{\kappa(e,b)d}$ different vectors $\iota \leq \varphi_{e,b}$.

We now study the splitting behaviour of job classes at inner nodes. Consider a set J' that is sent through an edge $f = \{u, v\}$ and then forth through an incident edge $g = \{v, w\}$. Then there are vectors ι_f and ι_g representing J' in the context of f and g respectively. However, some other set J'' represented by ι_f will also be represented by ι_g, that is ι_f translates uniquely into ι_g. We formalize this notion by the definition of a translation function $\tau_{f,g} : \{\iota \mid \iota \leq \varphi_{f,u}\} \to \{\iota' \mid \iota' \leq \varphi_{g,v}\}$. For each $i \in [\kappa(f, u)]$ there is a unique $i' \in [\kappa(g, v)]$ with $M_{f,v}(i) \cap M_{g,w} = M_{g,w}(i')$, i.e., the i-th connection type of $J_{f,u}$ translates into the i'-th connection type of $J_{g,v}$. Let $\upsilon_{f,g} : [\kappa(f, u)] \to [\kappa(g, v)]$ be given by $i \mapsto i'$. Now for each $\iota \leq \varphi_{f,u}$ we set $\tau_{f,g}(\iota) = (\iota'(1), \ldots, \iota'(\kappa(g, v)))$, with $\iota'(i') \in \mathbb{Z}^d_{\geq 0}$ and more precisely $\iota'(i') = \sum_{i \in \upsilon^{-1}_{f,g}(i')} \iota(i)$. With this we can formulate an analogue of (11). For a fix schedule σ let $\iota_{a,b}$ be the representative of the set of jobs that σ sends from a to b. Now let t be an inner node with neighbours u, v, w. For neighbours x, y of t, the set $\vec{J}_{x,t,y}$ considered in the last paragraph now has a representative in the contexts of $\{x, t\}$ and $\{t, y\}$. Fixing the first one $\iota_{x,t,y} \leq \iota_{x,t}$, the second one can be obtained via a transformation function, yielding:

$$\iota_{u,t} = \iota_{u,t,v} + \iota_{u,t,w} \qquad \iota_{t,u} = \tau_{(\{v,t\},\{u,t\})}(\iota_{v,t,u}) + \tau_{(\{w,t\},\{u,t\})}(\iota_{w,t,u}) \tag{14}$$

From now on we will use the up and down notion like before ($e = \{a, b\} \in E(T)$ with a above and b below). Let $\hat{\iota} \leq \varphi_{e,b}$ and $\check{\iota} \leq \varphi_{e,a}$ be candidate job classes to be send up and down e. Considering Lemma 19 we set $\mathrm{OPT}(e, \hat{\iota}, \check{\iota})$ to be the minimum value $\mathrm{OPT}(I_{e,t}(\hat{J}, \check{J}))$ where $\hat{J}$ is represented by $\hat{\iota}$ and $\check{J}$ by $\check{\iota}$.

For the case when b is a leaf not much changes. If $\eta^{-1}(b)$ is a job j^*, the class of $\{j^*\}$ has only one element and is represented by $\varphi_{e,b}$. Therefore we get that $\mathrm{OPT}(e, \hat{\iota}, \check{\iota}) = 0$ for $\hat{\iota} = \varphi_{e,b}$ and $\check{\iota} = 0$, and $\mathrm{OPT}(e, \hat{\iota}, \check{\iota}) = \infty$ otherwise. If $\eta^{-1}(b)$ is a machine i^*, we have $\varphi_{e,b} = 0$ and there are only two possible connection types for jobs from $J_{e,a}$, because jobs can be processed on i^* or not, i.e., $\kappa(e, a) \leq 2$. In any case we get $\mathrm{OPT}(e, \hat{\iota}, \check{\iota}) = \sum_{i \in [\kappa(e,a)]} p(\check{\iota}(i))$.

Now let b be an inner node again with lower neighbours ℓ and r to which it is connected via edges e_ℓ and e_r. We may assume that we know the values $\mathrm{OPT}(e_\ell, \hat{\lambda}, \check{\lambda})$ and $\mathrm{OPT}(e_r, \hat{\rho}, \check{\rho})$ for candidate job classes $(\hat{\lambda}, \check{\lambda}, \hat{\rho}, \check{\rho})$ to go up or down the left or right edge respectively. We want to identify the set $\Xi_e(\hat{\iota}, \check{\iota})$ of quadruples $(\hat{\lambda}, \check{\lambda}, \hat{\rho}, \check{\rho})$ that are compatible with $\hat{\iota}$ and $\check{\iota}$, i.e., that fulfil (14). For this let $\check{\iota}_\ell, \check{\iota}_r \leq \check{\iota}$ with $\check{\iota}_\ell + \check{\iota}_r = \check{\iota}$, $\hat{\lambda}_\ell, \hat{\lambda}_r \leq \varphi_{e_\ell,\ell}$ with $\hat{\lambda}_\ell + \hat{\lambda}_r \leq \varphi_{e_\ell,\ell}$, and $\hat{\rho}_\ell, \hat{\rho}_r \leq \varphi_{e_r,r}$ with $\hat{\rho}_\ell + \hat{\rho}_r \leq \varphi_{e_r,r}$, such that $\tau_{e_\ell,e}(\hat{\lambda}_\ell) + \tau_{e_r,e}(\hat{\rho}_r) = \hat{\iota}$. By setting $\hat{\lambda} = \hat{\lambda}_\ell + \hat{\lambda}_r$, $\check{\lambda} = \tau_{e,e_\ell}(\check{\iota}_\ell) + \tau_{e_r,e_\ell}(\hat{\rho}_\ell)$, $\hat{\rho} = \hat{\rho}_\ell + \hat{\rho}_r$ and $\check{\rho} = \tau_{e,e_r}(\check{\iota}_r) + \tau_{e_\ell,e_r}(\hat{\lambda}_r))$ we get such a tuple and the set $\Xi_e(\hat{\iota}, \check{\iota})$ is defined by such tuples.

Lemma 21. $\mathrm{OPT}(e, \hat{\iota}, \check{\iota}) = \min_{(\hat{\lambda},\check{\lambda},\hat{\rho},\check{\rho})} \max\{\mathrm{OPT}(e_\ell, \hat{\lambda}, \check{\lambda}), \mathrm{OPT}(e_r, \hat{\rho}, \check{\rho})\}$.

Proof. If the righthand side equals ∞, it is easy to see that the equation holds, and we will therefore assume $\mathrm{OPT}(e, \hat{\iota}, \check{\iota}) < \infty$. For given $\hat{\iota} \leq \varphi_{e,a}$ and $\check{\iota} \leq \varphi_{e,b}$ let $\check{J} \subseteq J_{e,a}$ be any set represented by $\check{\iota}$ and $\hat{J}^* \subseteq J_{e,b}$ an optimal one represented by $\hat{\iota}$, i.e., minimizing $\mathrm{OPT}(I_{e,b}(\hat{J}, \check{J}))$. Let σ^* be an optimal schedule for $I_{e,b}(\hat{J}, \check{J})$. Furthermore let $\hat{L}^*$, $\check{L}^*$, $\hat{R}^*$ and $\check{R}^*$ be the sets that σ^* sends up or down through e_ℓ or e_r respectively, and let $\hat{\lambda}^*$, $\check{\lambda}^*$,

$\hat{\rho}^*$ and $\check{\rho}^*$ their classes. Than σ^* induces schedules σ^*_ℓ and σ^*_{er} for $I_{e,\ell}$ and $I_{e,r}$. We get:

$$\begin{aligned} C_{\max}(\sigma^*) &= \max\{C_{\max}(\sigma^*_\ell), C_{\max}(\sigma^*_r)\} \\ &\geq \max\{\mathrm{OPT}(e_\ell, \hat{\lambda}^*, \check{\lambda}^*), \mathrm{OPT}(e_r, \hat{\rho}^*, \check{\rho}^*)\} \\ &\geq \min_{\substack{(\hat{\lambda},\check{\lambda},\hat{\rho},\check{\rho})\in \\ \Xi_e(\hat{\iota},\check{\iota})}} \max\{\mathrm{OPT}(e_\ell, \hat{\lambda}, \check{\lambda}), \mathrm{OPT}(e_r, \hat{\rho}, \check{\rho})\} \end{aligned}$$

Now let $(\hat{\lambda}, \check{\lambda}, \hat{\rho}, \check{\rho}) \in \Xi_e(\hat{\iota}, \check{\iota})$ minimizing the right hand side of of the considered equation with corresponding splitting vectors $\check{\iota}_\ell, \check{\iota}_r, \hat{\lambda}_\ell, \hat{\lambda}_r, \hat{\rho}_\ell, \hat{\rho}_r$. Moreover let $\hat{L}$ and $\hat{R}$ be optimal sets represented by $\hat{\lambda}$ and $\hat{\rho}$ respectively. Splitting $\hat{L}$, $\hat{R}$ and $\check{J}$ corresponding to the splitting of their job classes we can obtain sets $\check{L}$, $\check{R}$ and $\hat{J}$ that are represented by $\check{\lambda}$, $\check{\rho}$ and $\hat{\iota}$ respectively and fulfil (11). We have now $\mathrm{OPT}(e_\ell, \hat{\lambda}, \check{\lambda}) = \mathrm{OPT}(I_{e_\ell,\ell}(\hat{L}, \check{L}))$ and $\mathrm{OPT}(e_r, \hat{\rho}, \check{\rho}) = \mathrm{OPT}(I_{e_r,r}(\hat{R}, \check{R}))$. Let σ_ℓ and σ_r be respective optimal schedules. Than $\sigma := \sigma_\ell \cup \sigma_r$ is a schedule for $I_{e,b}(\hat{J}, \check{J})$ and $C_{\max}(\sigma)$ equals the right hand side of the considered equation. Since σ^* was chosen optimal with an optimal class representative we have furthermore $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. This yields:

$$\begin{aligned} \mathrm{OPT}(e, \hat{\iota}, \check{\iota}) &= C_{\max}(\sigma^*) = C_{\max}(\sigma) \\ &= \min_{\substack{(\hat{\lambda},\check{\lambda},\hat{\rho},\check{\rho})\in \\ \Xi_e(\hat{\iota},\check{\iota})}} \max\{\mathrm{OPT}(e_\ell, \hat{\lambda}, \check{\lambda}), \mathrm{OPT}(e_r, \hat{\rho}, \check{\rho})\} \end{aligned}$$

Moreover we get that $\hat{J}$ is optimal as well. □

Results. With these considerations a dynamic program for $P|M(j)|C_{\max}$ can be defined. This can be done in a way such that its running time is in $\mathcal{O}(m^2 n^{\mathcal{O}(d2^k)})$, proving Theorem 4 together with the Rounding Lemma and the considerations of Section 2.

Bi-cographs

We show that the path- and tree-hierarchical and nested cases are all special cases of the case that the incidence graph is a bi-cograph. Bi-cographs were introduced as a bipartite analogue of cographs [8].

Definition 22. For a bipartite graph $G = (A\dot{\cup}B, E)$ the *bi-complement* of G is the graph $(A\dot{\cup}B, \{\{a, b\} \mid a \in A, b \in B, \{a, b\} \notin E\})$. A graph is called *bi-cograph*, iff it is bipartite and can be reduced to isolated vertices by recursively bi-complementing its connected bipartite subgraphs.

It is known [9] that their cliquewidth and therefore also their rankwidth is bounded by 4. Furthermore by recursively bi-complementing the connected bipartite subgraphs, a certain decomposition of a given bi-cograph can be found in linear time that is similar to cotrees of cographs [8]. This decomposition can easily be turned into a branch decomposition, for which in the application studied here the number of connection types of jobs $\kappa(e, u)$ for every edge e of the decomposition and $v \in e$ is bounded by 2.

Lemma 23. *Let I be an instance of $P|M(j)|C_{\max}$ with path- or tree-hierarchical or nested restrictions. Then the incidence graph of I is a bi-cograph.*

Proof. We first consider the case that I has tree hierarchical restrictions. Let T be a corresponding rooted tree with $V(T) = \mathcal{M}$. Then there is at least one machine (the root of T) that can process all jobs. After bi-complementing the connected bipartite subgraphs of the incidence graph this machine is isolated. This can be repeated: After bi-complementing two more times the nearest descendants of the root in T that cannot process all jobs will be isolated. Iterating this, at some point all machines and therefore also all jobs will be isolated.

Now let I be an instance with nested restrictions. Note that the jobs $j \in \mathcal{J}$ with maximal $M(j)$ (with respect to $\subseteq$) are all in different connected components of the incidence graph and connected to all machines in their component. Hence they are isolated after bi-complementing the first time. If we bi-complement a second time and remove these jobs we get a new instance with nested restrictions and less jobs. By iterating this argument the claim follows. □

Acknowledgements. The Rounding Lemma in the presented form was formulated by Lars Rohwedder and Kevin Prohn as part of a student project.

References

[1] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] Yuichi Asahiro, Eiji Miyano, and Hirotaka Ono. Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. *Discrete Applied Mathematics*, 159(7):498–508, 2011.

[3] Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

[4] Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1):1–45, 1998.

[5] Derek G Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

[6] Tomáš Ebenlendr, Marek Krčál, and Jiří Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014.

[7] Leah Epstein and Asaf Levin. Scheduling with processing set restrictions: Ptas results for several variants. *International Journal of Production Economics*, 133(2):586–595, 2011.

[8] Vassilis Giakoumakis and Jean-Marie Vanherpe. Bi-complement reducible graphs. *Advances in Applied Mathematics*, 18(4):389–402, 1997.

[9] Vassilis Giakoumakis and Jean-Marie Vanherpe. Linear time recognition of weak bisplit graphs. *Electronic Notes in Discrete Mathematics*, 5:138–141, 2000.

[10] Petr Hlinenỳ and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing*, 38(3):1012–1032, 2008.

[11] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM (JACM)*, 23(2):317–327, 1976.

[12] Dušan Knop and Martin Kouteckỳ. Scheduling meets n-fold integer programming. *arXiv preprint arXiv:1603.02611*, 2016.

[13] Phokion G Kolaitis and Moshe Y Vardi. Conjunctive-query containment and constraint satisfaction. In *Journal of Computer and System Sciences*, 1998.

[14] Kangbok Lee, Joseph Y-T Leung, and Michael L Pinedo. A note on graph balancing problems with restrictions. *Information Processing Letters*, 110(1):24–29, 2009.

[15] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

[16] Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.

[17] Gabriella Muratore, Ulrich M Schwarz, and Gerhard J Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters*, 38(1):47–50, 2010.

[18] Jinwen Ou, Joseph Y-T Leung, and Chung-Lun Li. Scheduling parallel machines with inclusive processing set restrictions. *Naval Research Logistics (NRL)*, 55(4):328–338, 2008.

[19] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[20] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *Journal of Computer and System Sciences*, 76(2):103–114, 2010.

[21] Ola Svensson. Santa claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012.

[22] Stefan Szeider. On fixed-parameter tractable parameterizations of sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 188–202. Springer, 2003.

[23] Stefan Szeider. Not so easy problems for tree decomposable graphs. In *International Conference on Discrete Mathematics*, 2008.