Restricted Assignment Scheduling with Resource Constraints [*]

Gyorgy Dosa,[†] Hans Kellerer,[‡] Zsolt Tuza [§]

Abstract

We consider parallel machine scheduling with job assignment restrictions, i.e., each job can only be processed on a certain subset of the machines. Moreover, each job requires a set of renewable resources. Any resource can be used by only one job at any time. The objective is to minimize the makespan. We present approximation algorithms with constant worst-case bound in the case that each job requires only a fixed number of resources. For some special cases optimal algorithms with polynomial running time are given. If any job requires at most one resource and the number of machines is fixed, we give a PTAS. On the other hand we prove that the problem is APX-hard, even when there are just three machines and the input is restricted to unit-time jobs.

Keywords: scheduling; restricted assignment; resources; APX hardness; graph coloring.

[*] Research of G. Dosa and Zs. Tuza was supported in part by the National Research, Development and Innovation Office – NKFIH under the grant SNN 116095. G. Dosa was also supported in part by VKSZ_12-1-2013-0088 Development of cloud based smart IT solutions by IBM Hungary in cooperation with the University of Pannonia. All three authors are partially supported by Stiftung Aktion Österreich-Ungarn, under grant 92öu1.

[†] Department of Mathematics, University of Pannonia, H-8200 Veszprém, Egyetem u. 10, Hungary. E-mail: `dosagy@almos.vein.hu`

[‡] Institut für Statistik und Operations Research, Universität Graz, Universitätsstraße 15, 8010 Graz, Austria. E-mail: `hans.kellerer@uni-graz.at`

[§] Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences, H-1053 Budapest, Reáltanoda u. 13–15; and Department of Computer Science and Systems Technology, University of Pannonia, H-8200 Veszprém, Egyetem u. 10, Hungary. E-mail: `tuza@dcs.uni-pannon.hu`.

1 Introduction

We are given a set $J = \{1, \dots, n\}$ of n independent jobs that are to be scheduled on m parallel machines $M_1, \dots, M_m$. In the *Restricted Assignment problem* (RA-problem, for short) each job j can be executed on a specific subset $\mathcal{M}(j)$ of the machines, and on those machines the processing time of job j is p_j. The objective is to minimize the makespan. In the three field notation, we abbreviate this problem by $R|p_{ij} \in \{p_j, \infty\}|C_{\max}$.

Assume that additionally there are μ renewable resources $R_1, \dots, R_\mu$. Let Λ_k be the set of jobs which require resource R_k, $k = 1, \dots, \mu$, and let Λ_0 be the set of jobs which require no resources. Let λ_k denote the cardinality of set Λ_k, $k = 0, \dots, \mu$. Job j requires simultaneous availability of all resources in the set $\mathcal{R}(j) \subseteq \{R_1, \dots, R_\mu\}$ for processing; we denote by ρ_j the cardinality of $\mathcal{R}(j)$, $j = 1, \dots, n$. Any resource can be used by only one job at any time. It means that two jobs which require the same resource cannot be processed simultaneously. Note that in standard three-field notation problems with resources are classified by "$res\mu\sigma\rho$" which means that there are μ resources, the size of each resource does not exceed σ, and each job consumes no more than ρ units of a resource. Hence, our problem is abbreviated by $R|p_{ij} \in \{p_j, \infty\}, res\mu 11|C_{\max}$. In the following, we will call it *Restricted Assignment with Resources problem*, briefly RAR-problem. The *degree* of the problem is defined as the quantity $B = \max\limits_{j=1,\dots,n} \rho_j$, that is the maximum number of resources required by a job.

1.1 Related Problems

To the best knowledge of the authors, restricted assignment and resources were not considered previously together. The two types of conditions, however, have been studied separately.

The restricted assignment problem can be considered as a special case of the classical unrelated machine problem $R|\cdot|C_{\max}$, where job j on machine M_i has processing time p_{ij}. The paper of Lenstra et al. [24] contains a polynomial-time 2-approximation algorithm for $R|\cdot|C_{\max}$. On the negative side it is proven that one cannot get any worst-case ratio better than 3/2 for $R|\cdot|C_{\max}$ unless $\mathsf{P} = \mathsf{NP}$ holds. Ebenlendr et al. [14] extend this negative result even to the case where each set $\mathcal{M}(j)$ contains at most two machines.

Only for special cases of the RA-problem, small improvements of the 2-

approximation of [24] have been found recently. In [14] a 1.75-approximation is presented for the case that each job can be assigned to at most two machines. Chakrabarty et al. [12] consider the case where each job is either heavy ($p_j = 1$) or light ($p_j = \varepsilon$), for some parameter $\varepsilon > 0$. Their main result is a $(2 - \delta)$-approximate polynomial-time algorithm for a fixed constant $\delta > 0$.

There are many papers considering Multiprocessor Scheduling With Resources, MPSR for short. The first of them dealing with complexity analysis seems to be [18], another early one is [10]. The problem has several variants, some of them being of interest already on a single machine; see e.g. [25].

If each task requires at most one resource, i.e. $B = 1$, then MPSR with unit-time jobs admits a polynomial-time algorithm for an arbitrary number of processors, even with prescribed release times and deadlines [6]. On the other hand, for a variable number of resources the problem of minimizing maximum lateness becomes NP-hard, still with unit processing times of all the tasks, and already on just two processors [7]. Further related papers are [9, 22, 30]; see also the volume [8], the survey [15], and the references therein.

MPSR is equivalent to the problem of *Scheduling With Conflicts* (SWC), where we are given m parallel, identical machines, a set J of n jobs j with processing times p_j, and a conflict graph $G = (J, E)$. Each edge in E corresponds to a pair of jobs that cannot be scheduled concurrently. Hence, SWC can be considered as MPSR where each edge represents a resource which is required by the two jobs corresponding to that edge.[1] Vice versa, each instance of MSPR defines a unique conflict graph. Hence, each MSPR can be considered as SWC. The difference between the two models is that SWC classifies a problem according to the conflict graph, while in MPSR a problem is characterized by the resources required by the jobs, especially the number of resources is relevant. The latter model is much more realistic since most practical problems require only a constant number of resources.

The approach of MPSR, as compared to SWC, is also advantageous or more informative in the sense that in this way we embed the problem into a more general scenario, namely the MultiProfessor model, which we describe after the next paragraph.

Baker and Coffman [3] investigated SWC with unit-time jobs under the

[1] Since a resource may be required by more than two jobs, the minimum number of resources needed to transform an instance of SWC to that of MPSR is equal to the minimum number of complete subgraphs covering all edges of the conflict graph (if there are no additional restrictions).

name *mutual exclusion scheduling*. Even et al. [17] mainly investigated online algorithms for SWC, but also showed that the computation of the offline optimum of SWC is APX-hard, even for two machines and jobs whose processing times are integers between 1 and 4.

A further problem, introduced recently in [13], is called the *Multiprofessor Scheduling* problem. It involves a set $\mathcal{P} = \{P_1, \ldots, P_u\}$ of professors and a set $\mathcal{L} = \{L_1, \ldots, L_n\}$ of lectures (of specified durations), with two sets $\mathcal{C}$ and $\mathcal{C}^*$ of conditions given by pairs: $(P_i, L_j) \in \mathcal{C}$ means that professor P_i can deliver lecture L_j if it is assigned to him, while $(P_s, L_t)^* \in \mathcal{C}^*$ means that professor P_s has to be present when L_t is delivered by some *other* professor, who is assigned to this lecture.

Professors of the Multiprofessor Scheduling model correspond to machines of the RAR model; the lectures are the jobs, the duration of a lecture means the processing time. But RAR is only a particular case of the multiprofessor scheduling problem: distinction between machines and resources means a partition of professors into two classes: those only delivering lectures ('Professors'), and the others only attending ('Instructors').

1.2 Our Results

We prove inapproximability results and design approximation algorithms for the RAR-problem. Our main negative result is that the problem with unit-time jobs is APX-hard, already on three machines. In the case that each job requires only a bounded number of resources, we design approximation algorithms with constant worst-case bound, without any restrictions on processing times. For some special cases (e.g., unit-time jobs with degree $B = 1$) we design optimal algorithms with polynomial running time.

To derive the main negative result, we prove a theorem on graph coloring, which seems to be of interest on its own right, too. It states APX-hardness of the chromatic number on a restricted class of graphs. Interestingly enough, the corresponding result on scheduling was widely believed[2] to have been proved via triangle packing; but actually there is no quantitative equation between optimal triangle packings and optimal schedules. We illustrate this fact with a simple example in Figure 1. The vertices represent unit-time jobs,

[2] Section 1.2 of [17, p. 200] claims "a hardness gap for maximum packing of triangles in a graph (...) implies a hardness gap for scheduling with conflicts of unit jobs on 3 machines."

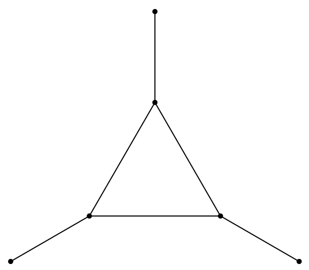

Figure 1: Small counterexample to a suspected relation between triangle packings and optimal schedules.

and the edges represent those pairs of jobs which are processable simultaneously. (The number of machines can be any $m \geq 3$.)

An optimal schedule requires makespan 3, by processing the two jobs of a pendant edge in each time slot. On the other hand, starting from the optimal triangle-packing and hence processing the three jobs of the central triangle at the same time, each of the remaining jobs (the three pendant vertices) require a further distinct time slot, yielding a schedule with makespan 4.

Section 2 is devoted to exact and approximation algorithms and their analysis, Section 3 gives a PTAS for the case of $B = 1$ with a fixed number of machines, and Section 4 contains inapproximability results. We finish in Section 5 with some conclusions and topics for future research.

1.3 Some Terminology

Given an instance of the RAR-problem, we use the term *assignment* if the jobs are assigned to the machines in a feasible way (i.e., the machine assignment restrictions are satisfied), without fixing the time slots to execute the jobs. If the jobs already got also their time slots where they are processed, this assignment is called *schedule*. For a set of jobs $J' \subset J$ we denote by $p(J')$ the total processing time of the jobs in J', i.e., $p(J') = \sum_{j \in J'} p_j$.

In a graph $G = (V, E)$ with vertex set V and edge set E, a set $S \subset V$ is *independent* if any two $v, w \in S$ are nonadjacent. The *independence number* of G is the maximum cardinality of an independent set in G. The *chromatic number* is the minimum number of independent sets whose union is the entire vertex set V. Equivalently, it is the minimum number of colors that can be assigned to the vertices in such a way that no two adjacent vertices get the same color. A *clique cover* of G is a collection of complete subgraphs whose union contains all vertices of G. Throughout this paper we restrict our attention to clique covers in which the complete subgraphs are mutually vertex-disjoint.

A *matching* in a graph is a collection of mutually vertex-disjoint edges; with an alternative terminology, any two edges of a matching are nonadjacent. The *matching number* is the maximum number of edges in a matching. The *chromatic index* of a graph or multigraph G is the minimum number of matchings whose union is the set E of all edges. Equivalently, it is the minimum number of colors that can be assigned to the edges in such a way that no two adjacent edges get the same color.

1.4 Approximation and L-Reduction

For an optimization problem P, if I is a problem instance and z is a solution on I, we denote by $\mathsf{Opt}_P(I)$ the optimum value on I. If not stated otherwise, let Opt_{RAR} denote the optimal solution value for the RAR-problem and Opt_{RA} be the optimal solution for the RA-problem, respectively. Let $\mathsf{val}_P(I, z)$ denote the actual value of solution z; we may simply write $\mathsf{val}(I, z)$ if P is understood. Given P and a real $\rho > 1$, an algorithm is a *ρ-approximation algorithm* if it determines a solution z for every instance I, with the property that $\mathsf{val}_P(I, z) \leq \rho\mathsf{Opt}_P(I)$ if P is a minimization problem, and $\mathsf{val}_P(I, z) \geq \frac{1}{\rho}\mathsf{Opt}_P(I)$ if P is a maximization problem. A family of ρ-approximation algorithms is called a *polynomial-time approximation scheme (PTAS)* if $\rho = 1+\varepsilon$ for any $\varepsilon > 0$ and the running time is polynomial with respect to the length of the problem input. A family of ρ-approximation algorithms is called a *fully polynomial-time approximation scheme (FPTAS)* if $\rho = 1 + \varepsilon$ for any $\varepsilon > 0$ and the running time is polynomial with respect to both the length of the problem input and $1/\varepsilon$.

Consider now two NP-optimization problems, say A and B. As introduced in [27], an *L-reduction* from A to B consists of two functions f and g, computable in polynomial time, for which there exist constants $\alpha, \beta > 0$ such that:

- if I is an instance of problem A, then $f(I)$ is an instance of problem B, satisfying
$$\mathsf{Opt}_{\mathrm{B}}(f(I)) \leq \alpha \cdot \mathsf{Opt}_{\mathrm{A}}(I)\,;$$

- if z is a solution on $f(I)$, then $g(z)$ is a solution on I, satisfying

$$|\mathsf{Opt}_{\mathrm{A}}(I) - \mathsf{val}_A(I, g(z))| \leq \beta \cdot |\mathsf{Opt}_{\mathrm{B}}(f(I)) - \mathsf{val}_B(f(I), z)|.$$

The problem class APX consists of those optimization problems P for which there exists a C-approximation with some constant C. The relevance of L-reduction is that if a problem A is APX-hard and there is an L-reduction form A to B then also B is APX-hard.

2 Exact and Approximation Algorithms

In this section we will present optimal algorithms with polynomial running time for two special cases, and a polynomial approximation algorithm for the general case. For the unit-time case with $B = 1$ we will give an exact algorithm with running time in $O((m^3 + m^2 n) \log n)$ and for the unit-time case with two machines we will give an optimal algorithm applying a maximum matching algorithm. For the general case we design a polynomial approximation algorithm with worst-case bound $B + 2 - 1/m$.

Solving the problem with two machines and unit-time jobs is straightforward using matching techniques.

Theorem 1. *The RAR-problem with unit-time jobs and two machines can be solved in polynomial time.*

Proof. We define a graph $G = (V, E)$. The vertices in V consist of the jobs $1, \ldots, n$. There is an edge between vertex i and vertex j if and only if jobs i and j have no resource in common which they require, i.e., $\mathcal{R}(i) \cap \mathcal{R}(j) = \emptyset$ and, moreover, it is not the case that $|\mathcal{M}(i)| = |\mathcal{M}(j)| = 1$ and $\mathcal{M}(i) = \mathcal{M}(j)$. In this graphical representation two unit-time jobs can be processed in the same time interval if and only if they are connected by an edge. If a maximum matching of this graph has ν edges, then the minimum makespan of a feasible schedule is $n - \nu$, and vice versa. Now the theorem follows by the fundamental result of Edmonds [16] that the matching number of graphs can be determined in polynomial time. □

The next theorem shows that for unit processing times and $B = 1$, the RAR-problem can be solved in polynomial time on any (maybe not even a constant) number of machines. Better bounds could be given for large m, but the main point here is polynomiality; moreover the formula is very strong if m is small, which is the typical case in scheduling.

Theorem 2. *The RAR-problem with n unit-time jobs and m machines can be solved in $O((m^3 + m^2 n) \log n)$ time for $B = 1$.*

Proof. Let f be a given positive integer. We will present an algorithm which examines in polynomial time whether an assignment exists with finish time less than or equal to f. The optimal value of f can then be found by binary search. Set $\lambda_{\max} = \max\{\lambda_1, \dots, \lambda_\mu\}$. Since $\lambda_{\max}$ is a lower bound for the optimal finish time, we may assume that $f \geq \lambda_{\max}$. Notice further that for $B = 1$, the sets $\Lambda_0, \Lambda_1, \dots, \Lambda_\mu$ are disjoint and hence $(\Lambda_0, \Lambda_1, \dots, \Lambda_\mu)$ is a partition of J. In the first stage, we will check whether there is a feasible assignment of jobs to machines with makespan at most f while ignoring the resource requirements. This is done by solving an appropriate network flow problem in a bipartite network N.

The network N is defined as follows. From a source node s we have arcs to all nodes of a set $V_1 = \{1, \dots, n\}$ of n nodes which represent the n jobs. The second part of the node set is given by $V_2 = \{M_1, \dots, M_m\}$. From node j there is an arc to node M_i if and only if $M_i \in \mathcal{M}(j)$. There is an arc from each node M_i to the sink t. All arcs have capacity 1 except for the arcs incident with t, which have capacity f. Applying standard techniques including the Flow Integrality Theorem, it follows that there is a feasible assignment of jobs to machines with makespan at most f if and only if N admits a flow from s to t with flow value n.

If a flow with flow value n exists, we take such an integral flow and use the computed assignment of jobs to machines in order to assign the jobs to time slots in the second stage. We have to assure that two jobs from the same set Λ_k, $k = 1, \dots, \mu$ are not processed simultaneously. Since there are no restrictions for the time slots for jobs in $\Lambda_{\mu+1}$, they are given into the time slots of idle time leftover after assigning the jobs from $\Lambda_1, \dots, \Lambda_\mu$. For the assignment of the jobs from $\Lambda_1, \dots, \Lambda_\mu$ we will define a bipartite multigraph G with vertex set $V = V_1 \cup V_2$. The set V_1 consists of μ vertices $R_1, \dots, R_\mu$. The vertex set V_2 corresponds to the m machines $M_1, \dots, M_m$. There is an edge between vertices R_k and M_i if and only if at least one job $j \in \Lambda_k$ has been assigned to M_i in the first stage. The multiplicity of the edge is equal to the number of jobs assigned to M_i from Λ_k.

Recall that the chromatic index of a (multi)graph G is the smallest number of colors needed to partition the edge set of G into color classes so that each class is a matching. Given such an edge coloring of G, we identify each color with a time slot; and then the λ_k jobs in Λ_k can be assigned arbitrarily to the λ_k different time slots corresponding the λ_k colors of the edges incident to R_k. It is easily seen that there is a feasible schedule of makespan f if we can find an edge coloring with at most f colors, and vice versa: "Each job requiring the same resource has to be processed in different time slots" translates into "All edges incident to some vertex $v \in V_1$ must receive different colors." "At each time, each machine executes at most one job" translates into "All edges incident to some $v \in V_2$ must receive different colors." It is well-known that a bipartite graph (or multigraph) with maximum degree h has also chromatic index h (see e.g. Berge [4, p. 247]).

Since $f \geq \lambda_{\max}$, G has maximum degree f, and hence the desired feasible schedule exists. To find such a schedule, we use the algorithm due to Alon for minimal edge coloring of bipartite multigraphs [2].

Let us look at the time complexity. The network N of Stage 1 has $|V_1| = n$ and the number of arcs in N is in $O(nm)$. Hence, the algorithm of Ahuja et al. [1] for bipartite network flows uses $O(m^3 + m^2 n)$ time. Since graph G has only n edges, we need $O(n \log n)$ time for the coloring algorithm in Stage 2, see [2]. Clearly, all jobs can be finished by time n. Thus, we can find a schedule with minimum makespan using binary search with at most $\log(n)$ steps. This gives the claimed time complexity. $\square$

We investigate now the RAR-problem with arbitrary processing times. Our approximation algorithm is again composed of two stages. In the first stage an assignment of jobs to machines with an approximation on the minimum finish time is given while ignoring the resources. This means that in the first stage a solution for the RA-problem is computed. In the second stage this assignment is transformed into a feasible schedule for the RAR-problem.

Let us now assume that we have found an assignment of the jobs to the machines using a heuristic H by ignoring the resources. Let $C_{\max}(H)$ denote the makespan produced by heuristic H. We fix in the second stage the schedule for each machine, taking into account that jobs requiring the same resource cannot be processed simultaneously. This is done by a greedy approach as follows.

Algorithm G

INPUT: An assignment of jobs to machines produced by a heuristic H

OUTPUT: A heuristic schedule S

1. Let t be the first time, such that some machine M_i is available and there is a job j, that may be started by M_i at time t. Select j to be the next job processed by M_i.
2. Repeat Step 1 until all jobs are scheduled. Stop.

Let $p(\Lambda_k)$ be the total length of all jobs which require resource R_k, i.e., $p(\Lambda_k) = \sum_{j\in\Lambda_k} p_j$, for $k = 1, \dots, \mu$. Set $p(\Lambda_{\max}) = \max_{k=1,\dots,\mu} p(\Lambda_k)$.

It can be easily seen that the running time of Algorithm G is at most $O(mn^2B)$. We now analyze its worst-case performance.

Theorem 3. *Let $C_{\max}(G)$ be the makespan of Algorithm S after applying heuristic H. Then*

$$C_{\max}(G) \leq C_{max}(H) + p(\Lambda_{\max}) \cdot B.$$

Proof. Let S be the schedule produced by G. Without loss of generality, suppose that machine M_m is the one which terminates schedule S. Let Q be the set of jobs processed by M_m after the last idle interval for M_m. We assume that Q is not empty, since otherwise $C_{\max}(G) = C_{max}(H)$ and the theorem obviously holds.

Each of the jobs of set Q requires at least one resource; otherwise a job from $\Lambda_{\mu+1}$ would have started earlier. Select one of the jobs $j \in Q$. It follows that the total idle time of M_m does not exceed the total time for processing all jobs that require at least one of the resources in $\mathcal{R}(j)$. Indeed, job j was not able to start in an idle interval because resources in $\mathcal{R}(j)$ were required by some other jobs in that entire interval. This means that the union of idle intervals for M_m is completely contained in the union of intervals in which jobs requiring resources from $\mathcal{R}(j)$ are processed. Thus, the total time of this union of intervals is bounded from above by $\sum\limits_{k\in\mathcal{R}(j)} p(\Lambda_k)$. Since $\rho_j \leq B$, we conclude that

$$C_{\max}(G) \leq C_{max}(H) + \sum_{k\in\mathcal{R}(j)} p(\Lambda_k) \leq C_{max}(H) + p(\Lambda_{\max}) \cdot B.$$

$\square$

Corollary 4. *If heuristic H provides an α-approximation for the RA-problem, we obtain an $(\alpha + B)$-approximation algorithm for the RAR-problem.*

Proof. Recall that Opt_{RAR} denotes the optimal solution value for the RAR-problem and Opt_{RA} the optimal solution for the RA-problem, respectively. Then, we have

$$\mathsf{Opt}_{RAR} \geq \max\{\mathsf{Opt}_{RA}, p(\Lambda_{\max})\}.$$

Using Theorem 3 we get

$$C_{\max}(S) \leq \alpha \cdot \mathsf{Opt}_{RA} + p(\Lambda_{\max}) \cdot B \leq (\alpha + B) \cdot \mathsf{Opt}_{RAR}.$$

□

Theorem 5. *There is a polynomial-time $(2-1/m+B)$-approximation algorithm for the RAR-problem on m machines with arbitrary processing times.*

Proof. Consider the scheduling problem of minimizing the makespan on unrelated machines, $R|\cdot|C_{\max}$. In the unrelated machine problem, job j has processing time p_{ij} on machine i.

For an assignment of jobs to machines in the RA-problem the processing times p_{ij} reduce to

$$p_{ij} = \begin{cases} p_j, & i \in \mathcal{M}(j) \\ \infty, & \text{otherwise.} \end{cases}$$

Recall that Lenstra et al. [24] present a 2-approximation algorithm for problem $R|\cdot|C_{\max}$. Using the improved rounding scheme of Shchepin and Vakhania [29] it is even possible to find a solution for IP with makespan

$$C_{\max}(SV) \leq \left(2 - \frac{1}{m}\right) \mathsf{Opt}_{RA}, \tag{1}$$

where $C_{\max}(SV)$ denotes the makespan obtained by applying the algorithm in [29] and Opt_{RA} corresponds to an optimal solution of the RA-problem. Replacing α by $2 - 1/m$ in Corollary 4 we get the desired result. □

Corollary 6. *If B is a constant, there is a polynomial-time approximation algorithm with constant worst-case bound for the RAR-problem.*

Corollary 7. *There is a polynomial-time $(1 + B)$-approximation algorithm for the RAR-problem with unit-time jobs and abritrary number of machines.*

Proof. It can be easily seen that the RA-problem with unit-time jobs is solvable in polynomial time by maximum cardinality bipartite matching. □

Corollary 8. *For any fixed $\epsilon > 0$ there is a polynomial-time $(1+\varepsilon+B)$-approximation algorithm for the RAR-problem, when the number of machines m is a constant.*

Proof. It is well-known that there is an FPTAS for the problem $Rm|\cdot|C_{\max}$ with a fixed number of machines. $\square$

A special case of the restricted assignment problem is the problem where the sets $\mathcal{M}(j)$ are *nested*: for any two jobs j and k, either $\mathcal{M}(j) \cap \mathcal{M}(k) = \emptyset$, or $\mathcal{M}(j) \subseteq \mathcal{M}(k)$, or $\mathcal{M}(k) \subseteq \mathcal{M}(j)$. Muratore et al. [26] derive a polynomial-time approximation scheme for the restricted assignment problem with nested constraints (without any assumptions on m). This results in the following corollary.

Corollary 9. *For any fixed $\epsilon > 0$ there is a polynomial-time $(1+\varepsilon+B)$-approximation algorithm for the RAR-problem with nested constraints, even when the number of machines m is part of the input.*

Proof. Replace α by $1+\varepsilon$ for this special RA-problem. $\square$

3 A PTAS for a Fixed Number of Machines and B = 1

In this section we present a polynomial time approximation scheme (PTAS) for the RAR-problem with $B = 1$ for which the number of machines m is fixed. Our proof is an extension of the proof in [20] for the problem with parallel dedicated machines with a single resource $PDm|res111|C_{\max}$ where each job has to be processed on exactly one machine and one resource is available. For the sake of completeness, we will repeat those parts of the proof from [20] that are slightly modified but necessary for understanding.

Recall from the preceding section that $C_{\max}(SV)$ denotes the makespan obtained by applying the algorithm in [29] to the RA-problem. Then, define

$$C = \max\{p(\Lambda_{\max}), p(J)/m, C_{\max}(SV)/2\}.$$

Here, $p(\Lambda_{\max})$ and $p(J)/m$ are obvious lower bounds for Opt_{RAR}. We conclude for $B = 1$ from (1) that

$$C \le \mathsf{Opt}_{RAR}. \tag{2}$$

Applying Theorem 3 with $C_{\max}(SV)$ we get $\mathsf{Opt}_{RAR} \le C_{\max}(SV) + p(\Lambda_{\max})$ and thus

$$\mathsf{Opt}_{RAR} \le 3C. \tag{3}$$

Let $\varepsilon > 0$ be a small number and set $\tilde{\varepsilon} = \varepsilon/74$. We introduce the sequence of real numbers $\delta_1, \delta_2, \ldots$ such that

$$\delta_t = \left(\frac{\tilde{\varepsilon}}{m^3}\right)^{2^t}.$$

For each integer t, $t \ge 1$, define the set of jobs $J^t = \{j \in J \mid \delta_t^2 C < p_j \le \delta_t C\}$. Obviously, $J^1, J^2, \ldots$ are mutually disjoint and their union is J. Thus, there is a positive integer $t_0 \le \lceil \frac{m}{\tilde{\varepsilon}} \rceil$ such that $p(J^{t_0}) \le \tilde{\varepsilon} p(J)/m \le \tilde{\varepsilon} C$ holds. Choose $\delta = \delta_{t_0}$. Partition the set J of jobs into the set of *big jobs* J_1, the set of *medium jobs* J_2, and the set of *small jobs* J_3, defined as follows:

$$\begin{aligned} J_1 &= \{j \mid \delta C < p_j\}, \\ J_2 &= \{j \mid \delta^2 C < p_j \le \delta C\}, \\ J_3 &= \{j \mid p_j \le \delta^2 C\}. \end{aligned} \tag{4}$$

The corresponding cardinalities of J_1, J_2, J_3 are given by n_1, n_2, n_3, respectively. Note that by definition $J_2 = J^{t_0}$, so that

$$p(J_2) \le \tilde{\varepsilon}\, \mathsf{Opt}_{RAR}. \tag{5}$$

Besides,

$$\frac{\tilde{\varepsilon}}{m^3} \ge \delta \ge \left(\frac{\tilde{\varepsilon}}{m^3}\right)^{2^{\lceil \frac{m}{\tilde{\varepsilon}} \rceil}}. \tag{6}$$

We get from (3) that

$$n_1 < \frac{3}{\delta} m. \tag{7}$$

Consider the set of *restricted schedules* $\mathcal{S}_{\mathcal{R}}$ which consists of all the schedules, in which the starting times of the big jobs are non-negative integer multiples of $\delta^2 C$. Let $\mathsf{Opt}_{\mathcal{S}_{\mathcal{R}}}$ denote the optimal solution value for the RAR-problem restricted to class $\mathcal{S}_R$. Assume that the schedules are represented by Gantt-charts where the horizontal axis corresponds to the time. We transform an optimal schedule S^* of the RAR-problem into a schedule $S_R \in \mathcal{S}_{\mathcal{R}}$ by shifting jobs iteratively to the right, beginning with the smallest starting time of a big job which is not equal to a multiple of $\delta^2 C$. In each iteration jobs are shifted to the right by at most $\delta^2 C$ and after n_1 iterations we obtain a schedule of class $\mathcal{S}_{\mathcal{R}}$. Using (3) and (7) we get

$$\mathsf{Opt}_{\mathcal{S}_{\mathcal{R}}} \leq \mathsf{Opt}_{RAR} + 3m\delta C \leq 3C\,(1+m\delta)\,. \tag{8}$$

For the jobs of set J_1, define $\mathcal{S}_{\mathcal{B}}$ as the class of schedules which contains all possible schedules of big jobs with starting and completion times in $[0; 3C\,(1+m\delta)]$ such that all starting times are non-negative integer multiples of $\delta^2 C$. Consider an arbitrary schedule $S_B \in \mathcal{S}_{\mathcal{B}}$. Let $\tau_1 < \tau_2 < \ldots < \tau_q$ be the increasing sequence of all positive distinct starting and completion times of the big jobs. By (7) we have

$$q < \frac{6}{\delta}\, m. \tag{9}$$

For schedule S_B, define the following sequence of time intervals

$$I_0 = [0;\tau_1], I_1 = [\tau_1;\tau_2], \ldots, I_h = [\tau_h;\tau_{h+1}], \ldots, I_{q-1} = [\tau_{q-1};\tau_q], I_q = [\tau_q;\infty[.$$

Clearly, inside a time interval I_h, $0 \leq h \leq q$, no big job starts or completes. With $\tau_{q+1} = \infty$ and $\tau_0 = 0$ denote the *length* of I_h by $\ell(I_h) = \tau_{h+1} - \tau_h$ for $h = 0, \ldots, q-1$ and $\ell(I_q) = \infty$.

When a time interval I_h is associated with a machine M_i we denote it by $I_{i,h}$ and define its *capacity* by

$$c(I_{i,h}) = \begin{cases} \ell(I_h), & \text{if no big job is processed on } M_i \text{ in interval } I_h \\ 0, & \text{otherwise.} \end{cases}$$

The *capacity for resource* R_k *of interval* I_h is given as

$$r_k(I_h) = \begin{cases} \ell(I_h), & \text{if no big job of } \Lambda_k \text{ is processed in interval } I_h \\ 0, & \text{otherwise.} \end{cases}$$

The value $r_k(I_h)$ specifies the maximum total processing time of small or medium jobs in Λ_k which can be processed during time interval I_h. As in the proof of Theorem 5 the processing times p_{ij} reduce to

$$p_{ij} = \begin{cases} p_j, & i \in \mathcal{M}(j) \\ \infty, & \text{otherwise.} \end{cases}$$

We want to assign the small jobs *preemptively* into schedule S_B. For this purpose the linear program $LP(S_B)$ is defined as follows:

$$\begin{aligned}
&\text{minimize } z \\
\text{s.t.} \quad & \sum_{j\in J_3} x_{(i,h),j}\, p_{i,j} \le c(I_{i,h}), \qquad i=1,\dots,m, \quad h=0,\dots,q-1, && (10)\\
& \sum_{i=1}^{m}\sum_{j\in \Lambda_k} x_{(i,h),j}\, p_{i,j} \le r_k(I_h), \qquad k=1,\dots,\mu, \quad h=0,\dots,q-1, && (11)\\
& \sum_{j\in J_3} x_{(i,q),j}\, p_{i,j} \le z, \qquad i=1,\dots,m, && (12)\\
& \sum_{i=1}^{m}\sum_{j\in \Lambda_k} x_{(i,q),j}\, p_{i,j} \le z, \qquad k=1,\dots,\mu, && (13)\\
& \sum_{h=0}^{q}\sum_{i=1}^{m} x_{(i,h),j} = 1, \qquad j\in J_3, && (14)\\
& x_{(i,h),j} \ge 0.
\end{aligned}$$

The variable $x_{(i,h),j}$ determines the rate of small job j which is processed in interval I_h on machine M_i. Hence, (14) guarantees that job j is fully distributed among the machines and that $0 \le x_{(i,h),j} \le 1$. Inequalities (10) ensure that the total processing time of small jobs executed in interval I_h on machine M_i does not exceed the interval length $\ell(I_h)$. Inequalities (11) guarantee that the total processing time of jobs in Λ_k executed in interval I_h does not exceed $\ell(I_h)$. By (12) and (13) the optimal objective function value z is not smaller than the maximum of the total processing time of small jobs processed on a machine after time τ_q and the maximum of the total processing time of small jobs which require a certain resource. Notice that $LP(S_B)$ only assigns small jobs to time intervals, but no starting times of the jobs are given and a preempted job can be assigned to different machines.

The linear program $LP(S_B)$ consists of $m(q+1)$ inequalities (10), (12), $\mu(q+1)$ inequalities (11), (13) and n_3 equations (14). Let an optimal basic solution of $LP(S_B)$ with optimal solution value z^* be represented as a matrix

$$\Gamma = \left(\gamma_{(i,h),j}\right), \quad i = 1, \dots, m,\ h = 0, \dots, q,\ j \in J_3.$$

The rows of Γ correspond to all possible pairs (i, h) taken in any order, the n_3 columns correspond to different small jobs. The values v_1 and v_2 denote the number of inequality constraints (10), (12) and (11), (13), respectively, which are satisfied as equalities by Γ. By elementary linear programming theory, the number γ_+ of strictly positive values in Γ is at most $n_3 + v_1 + v_2$. Clearly, $v_1 \leq m(q+1)$. For $\mu \leq m$, we have $v_2 \leq m(q+1)$ as well. If $\mu > m$, then for each h, $0 \leq h \leq q$, inequalities (11) and (13) hold as equality for at most m resource values k. We conclude that at most $m(q+1)$ inequality constraints (11), (13) are non-redundant. Thus, we have shown that

$$\gamma_+ \leq n_3 + 2m(q+1). \tag{15}$$

Call positive solution values $\gamma_{(i,h),j}$ with $\gamma_{(i,h),j} < 1$ *split values*; otherwise, call positive solution values with $\gamma_{(i,h),j} = 1$ *non-split values*. Notice that if $\gamma_{(i,h),j}$ is a non-split value, then job j is completely assigned to time interval I_h on machine M_i. Partition the set of small jobs J_3 into the set J_3^s of *small split jobs*, i.e., those jobs j for which there is a row (i, h) such that $0 < \gamma_{(i,h),j} < 1$, and into the set J_3^{ns} of *small non-split jobs*, i.e. those jobs j for which there is a row (i, h) such that $\gamma_{(i,h),j} = 1$.

Since each job is processed on at least one machine and in at least one time interval, it follows that each of the n_3 columns of matrix Γ has at least one positive entry. By (15) there are at most $2m(q+1)$ columns with more than one positive entry, and the positive entries of each such column correspond to the split values of Γ. Hence, there are at most $4m(q+1)$ split values and at most $2m(q+1)$ small split jobs. Since the maximum processing time of a small split job is $\delta^2 C$, we obtain

$$p(J_3^s) \leq 2m(q+1)\delta^2 C. \tag{16}$$

Temporarily, ignore the set J_3^s. For each time interval I_h we define an open shop problem where the small non-split jobs which use the same resource and are assigned to I_h by $LP(S_B)$ form an operation. The schedule of each such open shop problem corresponds to time slots to which we assign the original jobs afterwards. More precisely, let $J_{(i,h),\Lambda_k}$ denote the set of small non-split jobs which require resource k and which are assigned by $LP(S_B)$ to machine M_i in time interval I_h, i.e.,

$$J_{(i,h),\Lambda_k} = \left\{ j \in J_3^{ns} \mid j \in \Lambda_k, \gamma_{(i,h),j} = 1 \right\} \quad i = 1, \dots, m,\ h = 0, \dots, q,\ k = 1, \dots, \mu.$$

Analogously,

$$J_{(i,h),\Lambda_0} = \left\{ j \in J_3^{ns} \mid j \in \Lambda_0, \gamma_{(i,h),j} = 1 \right\}$$

is the set of small non-split jobs which require no resource and which are assigned by $LP(S_B)$ to machine M_i in time interval I_h. Then, define for each time interval I_h, $h = 0, \dots, q$, an open shop problem $OS(h)$ on m machines with $m + \mu$ jobs as follows. There are given μ jobs $O_{h,k}$, $k = 1, \dots, \mu$, with operations $O_{(i,h),k}$ on machine M_i, $i = 1, \dots, m$. These operations have the processing times

$$p(O_{(i,h),k}) = \sum_{j \in J_{(i,h),\Lambda_k}} p_j, \quad i = 1, \dots, m,\ k = 1, \dots, \mu. \tag{17}$$

Furthermore, there are m jobs $O_{(i,h),0}$, $i = 1, \dots, m$, which consist of single operations, also denoted as $O_{(i,h),0}$, which have to be processed on machine M_i. They have the processing times

$$p(O_{(i,h),0}) = \sum_{j \in J_{(i,h),\Lambda_0}} p_j, \quad i = 1, \dots, m. \tag{18}$$

Each job $O_{h,k}$ of problem $OS(h)$, $h = 0, \dots q$, has release time τ_h. We allow the jobs to be preemptive. Hence, the problem can be classified in 3-field notation as $Om|\text{pmtn}|C_{\max}$. Let $\mathsf{Opt}_{OS(h)}$ denote the optimal solution of problem $OS(h)$.

For a classic open shop problem on m machines with n jobs and operation times q_{ij}, $i = 1, \dots, m$, $j = 1, \dots n$, there are two obvious lower bound on the optimal makespan:

$$LB_1 = \max \left\{ \sum_{j=1}^{n} q_{ij} \mid i = 1, \dots, m \right\} \quad \text{(machine based lower bound)},$$

and

$$LB_2 = \max \left\{ \sum_{i=1}^{m} q_{ij} \mid j = 1, \dots, n \right\} \quad \text{(job based lower bound)}.$$

If the open shop problem is preemptive, there is a polynomial algorithm which finds an optimal schedule with solution value equal to $\max\{LB_1, LB_2\}$ (see [28] for details). Inequalities (10) and (11) imply that

$$\mathsf{Opt}_{OS(h)} \leq \tau_{h+1}, \quad h = 0, \ldots, q-1. \tag{19}$$

Analogously, inequalities (12) and (13) correspond to the machine based and the job based lower bound, respectively. This implies that

$$\mathsf{Opt}_{OS(q)} \leq \tau_q + z^*. \tag{20}$$

Notice that there is an optimal schedule for $Om|\text{pmtn}|C_{\max}$ that has no more than $4m^2$ preemptions. A possible algorithm for finding such a schedule is described in [23].

Consider such an optimal schedule for problem $OS(h)$, $h = 0, \ldots, q$. For each k, $k = 0, \ldots, \mu$, and each machine M_i, $i = 1, \ldots, m$, identify the sequence $T^u_{(i,h),k}$, $u = 1, 2, \ldots,$ of time slots in which the operation $O_{(i,h),k}$ is processed (preemptively) on machine M_i. For each such interval $T^u_{(i,h),k}$ determine its length $\ell(T^u_{(i,h),k})$. Notice that,

$$p(O_{(i,h),k}) = \sum_u \ell\left(T^u_{(i,h),k}\right). \tag{21}$$

We are now able to formulate the PTAS for our problem.

Algorithm *RARm*

INPUT: An instance of the RAR-problem with $B = 1$, a fixed number of machines m and accuracy $\varepsilon > 0$

OUTPUT: A heuristic schedule S_ε

1. For a given ε, determine $\tilde{\varepsilon}$ and δ. Partition the set J of jobs into the subsets J_1, J_2 and J_3 of big, medium and small jobs, respectively, as defined in (4).

2. For each schedule $S_B \in \mathcal{S_B}$ solve the corresponding linear program $LP(S_B)$ and find the optimal value z^* of the objective function. Identify schedule $S^*_B \in \mathcal{S_B}$ for which the value $z^* + \tau_q$ attains its minimum. Denote $I_{q+1} = [\tau_q; \tau_q + z^*]$.

3. For the schedule S_B^* solve the corresponding $q+1$ open shop problems $OS(0), OS(1), \ldots, OS(q)$ such that each optimal solution has at most $4m^2$ preemptions. For each $i = 1, \ldots, m$, $h = 0, \ldots, q$, $k = 0, \ldots, \mu$, assign the jobs in $J_{(i,h),\Lambda_k}$ non-preemptively to the corresponding time-slots $T^u_{(i,h),k}$, $u = 1, 2, \ldots$, by using First Fit. The small non-split jobs which could not be sequenced are collected in set J_3^{ns1}.

4. The jobs which are not assigned yet, i.e., the medium jobs J_2, small split jobs in J_3^s and the small non-split jobs in J_3^{ns1}, are sequenced at the end of the schedule in any order. We obtain schedule S_ε with makespan $C_{\max}(S_\varepsilon)$.

Theorem 10. *Algorithm RARm is a PTAS for the RAR-problem with a fixed number of machines and* $B = 1$*.*

Proof. Algorithm *RARm* runs in polynomial time for fixed m and ε. This is guaranteed by the fact that the maximum number of schedules in $\mathcal{S}_{\mathcal{B}}$ is in $O((\frac{m}{\delta^2})^{\frac{3m}{\delta}})$ and is therefore a constant. We prove that the algorithm produces a schedule with makespan not greater than $(1+\varepsilon)\mathsf{Opt}_{RAR}$.

The optimal solution of $LP(S_B^*)$ gives a lower bound for the optimal solution values of schedules in $\mathcal{S}_{\mathcal{R}}$, i.e.,

$$z^* + \tau_q \leq \mathsf{Opt}_{\mathcal{S}_{\mathcal{R}}}. \tag{22}$$

Inequalities (19), (20) guarantee that the time-slots produced in Step 3 fit into the intervals $I_0, I_1, \ldots, I_{q+1}$. Therefore, the small jobs assigned in Step 3 are finished before $\tau_q + z^*$ and the makespan of schedule S_ε is bounded by the total processing time of the jobs assigned in Step 4 plus $\tau_q + z^*$, i.e.,

$$C_{\max}(S_\varepsilon) \leq \tau_q + z^* + p(J_2) + p(J_3^s) + p\left(J_3^{ns1}\right). \tag{23}$$

The total processing time of the medium jobs and the small split jobs is bounded by (5) and (16). We have to estimate the number of small non-split jobs that cannot be scheduled in Step 3 of the algorithm. Therefore, we turn to the time slots $T^u_{(i,h),k}$ introduced in Step 3 of Algorithm *RARm*. Collect the time slots $T^u_{(i,h),k}$ which correspond to the same operation in set $\hat{T}_{(i,h),k}$, i.e.,

$$\hat{T}_{(i,h),k} = \left\{T^u_{(i,h),k} \middle| \, u = 1, 2, \ldots\right\}.$$

First, consider the time slots which belong to one-element sets $\hat{T}_{(i,h),k}$. From (17), (18) and (21) follows that in Step 3 of the algorithm all jobs of $J_{(i,h),\Lambda_k}$ are processed on machine M_i as a single block.

Collect the time slots which belong to sets $\hat{T}_{(i,h),k}$ with more than one element, in set $\hat{T}_1$ and define $t_1 = |\hat{T}_1|$. Recall that there are at most $4m^2$ preemptions for each open shop problem $OS(h)$, $h = 0, \dots, q$. Any set $\hat{T}_{(i,h),k}$ with v elements ($v \geq 1$) corresponds to $v - 1$ preemptions in interval I_h. Hence, we get $t_1 \leq 8m^2(q+1)$ and at most $8m^2(q+1)$ small non-split jobs cannot be assigned by First Fit in Step 3. This results in

$$p\left(J_3^{ns1}\right) \leq 8m^2(q+1)\delta^2 C. \tag{24}$$

Now it is possible to estimate $C_{\max}(S_\varepsilon)$ by inserting (5), (16) and (24) in (23). We get

$$C_{\max}(S_\varepsilon) \leq \tau_q + z^* + \tilde{\varepsilon}\,\mathsf{Opt}_{RAR} + 2m(q+1)\delta^2 C + 8m^2(q+1)\delta^2 C. \tag{25}$$

Applying (2), (8) and (22) inequality (25) simplifies to

$$C_{\max}(S_\varepsilon) \leq \left(1 + 3m\delta + \tilde{\varepsilon} + 2m(q+1)\delta^2 + 8m^2(q+1)\delta^2\right)\,\mathsf{Opt}_{RAR}.$$

Inserting (9) and afterwards (6) we deduce

$$\begin{aligned} C_{\max}(S_\varepsilon) &\leq \left(1 + 3m\delta + \tilde{\varepsilon} + 12m^2\delta + 2m\delta^2 + 48m^3\delta + 8m^2\delta^2\right)\mathsf{Opt}_{RAR} \\ &\leq \left(1 + \frac{3\tilde{\varepsilon}}{m^2} + \tilde{\varepsilon} + \frac{12\tilde{\varepsilon}}{m} + \frac{2\tilde{\varepsilon}^2}{m^5} + 48\tilde{\varepsilon} + \frac{8\tilde{\varepsilon}^2}{m^4}\right)\mathsf{Opt}_{RAR} \\ &\leq (1 + 74\tilde{\varepsilon})\,\mathsf{Opt}_{RAR} \\ &= (1 + \varepsilon)\,\mathsf{Opt}_{RAR}. \end{aligned}$$

Thus, we have shown that Algorithm $RARm$ is a PTAS. The theorem is proved. □

4 Inapproximability for Unit-Time Jobs

In this section we study the RAR-problem for unit-time jobs, but with no upper bound on B. We first show that this subproblem cannot be approximated within any constant bound. Then, by studying graph coloring on a restricted class, we will prove that even the problem with unit-time jobs and just three machines is APX-hard. Notice that for the unrelated machine problem with any fixed number of machines, namely $Rm|\cdot|C_{\max}$, an FPTAS can be constructed by using standard dynamic programming techniques (see e.g. [19]).

Let us note further that the RAR-problem with any fixed number m of machines and arbitrary processing times is in APX, because a trivial m-approximation is to schedule all jobs sequentially (each job j on any machine chosen from $\mathcal{M}(j)$). Hence, if m is bounded, APX-hardness and APX-completeness are equivalent. For unbounded m, however, approximation is much less efficient, as shown in the following result. It extends the corresponding observation from [17, p. 209] to a stronger form allowed by the current model; it is adjusted from [13, Theorem 11] to the RAR terminology.

Theorem 11. *The RAR-problem cannot be approximated within $O(n^{1-\epsilon})$ for any fixed $\epsilon > 0$ in polynomial time, unless* $\mathsf{P} = \mathsf{NP}$*, even when restricted to instances where all jobs have unit time, each job can be processed on exactly one machine, each machine processes exactly one job, and each resource is required by at most two jobs.*

Proof. We recall a reduction from [13], where it was constructed for a more general problem, but works also for the current restricted one, too, without modification. We apply the theorem of Zuckerman [31], which states that the chromatic number of graphs does not admit a polynomial-time $O(n^{1-\epsilon})$-approximation for any $\epsilon > 0$, where n is the number of vertices. So, let $G = (V, E)$ be a graph of order n, and denote its vertices by $v_1, \ldots, v_n$. Each v_i corresponds to a unit-time job i. We assume here $m = n$, and that machine M_i is dedicated to process job i. In this way the machines can work independently at any time, except that restrictions will be created by the resource requirements.

We define $\mu = |E|$, and index the resources with the edges of G. If $v_i v_j \in E$, then there is a resource R_{ij} required for both jobs i and j (but not for any other job). This ensures that i and j cannot be processed at the same time.

Since there are no more restrictions, it follows that a subset $S \subset \{1, \ldots, n\}$ of jobs can be processed simultaneously in the same time slot if and only if the set $\{v_i \mid i \in S\}$ is independent in G. Thus, minimum makespan equals the chromatic number of G. $\square$

The scheduling problem for parallel dedicated machines under resource constraints, briefly $PD|res\mu 11|C_{\max}$, is a special case of the RAR-problem where each job has to be processed on exactly one machine. It was shown in [21] that there is a PTAS for this problem when the number of machines is a constant. In contrast, Theorem 11 shows that even for unit-time jobs it is non-approximable with a constant worst-case bound when the number of machines is part of the input.

Corollary 12. *The scheduling problem $PD|res\mu 11|C_{\max}$ cannot be approximated within $O(n^{1-\epsilon})$ for any fixed $\epsilon > 0$ in polynomial time even for unit-time jobs, unless $\mathsf{P} = \mathsf{NP}$.*

It should be noted that the problem instances constructed in the proof above have input sizes of order n^2. Currently we do not know what kind of lower bound holds for instances whose size grows more slowly, e.g. is linear in n.

As the main tool for the APX-hardness result, next we prove a theorem on graph colorings and clique covers, which is of interest on its own right, too. The algorithmic problems Chromatic Number and Minimum Clique Cover take a graph G as input, and ask for the chromatic number of G and the minimum number of complete subgraphs[3] in a clique cover of G, respectively. Although there is an extensive literature on the approximability of the chromatic number of graphs, we were not able to find the theorem given below.

Theorem 13. *The following optimization problems are* APX*-complete:*

(*i*) *the* Chromatic Number *problem restricted to graphs of independence number 3,*

(*ii*) *the* Minimum Clique Cover *problem restricted to graphs whose clique number is 3,*

(*iii*) *even more restrictively the* Minimum Clique Cover *problem on graphs whose clique number is 3 and maximum degree is 4.*

[3] Since all induced subgraphs of a complete graph are complete, the minimum is the same independently of whether it is assumed that the subgraphs used in the cover are vertex-disjoint.

Proof. Membership in APX is immediately seen, because the optimum clearly is at most n and at least $n/3$ for every graph of order n; thus, the solution 'n' is a trivial 3-approximation. Also, equivalence between (i) and (ii) holds by taking complementary graphs. For this reason it suffices to prove that there is no PTAS for Minimum Clique Cover under the degree-4 condition, unless $\mathsf{P} = \mathsf{NP}$.

We apply L-reduction from the Max-2-SAT-3 problem, which is APX-hard by the theorem of Berman and Karpinski [5]. (More explicitly, if $\mathsf{P} \neq \mathsf{NP}$, then it is not possible to approximate the optimum within a multiplicative 1.0005 in polynomial time.) The construction below is quite similar to the one given by Caprara and Rizzi [11] for the problem of determining the maximum number of vertex-disjoint triangles in graphs,[4] but there are substantial differences: the current structure is somewhat simpler, but nevertheless its verification is much more tedious.

Let Φ be a Boolean formula over the n variables $x_1, \ldots, x_n$ with m clauses $c_1, \ldots, c_m$ such that each clause is either a single literal or the disjunction of two literals, and each variable occurs in at most three clauses (where both literals x_i and $\neg x_i$ are counted). The task is to maximize the number of satisfied clauses in a truth assignment. Note that

$$\mathsf{Opt}(\Phi) \geq m/2$$

because the all-true and all-false assignments together satisfy each clause, therefore one of them satisfies at least half of the clauses.

We may assume without loss of generality that each variable x_i occurs in at least one positive and also in at least one negative literal, for otherwise we can set all clauses containing x_i true, hence reducing the instance to a smaller one in constant time. Thus the total number of literals is at least $2n$; but it cannot exceed $2m$ (by the condition on clause size), therefore $m \geq n$ holds, which implies

$$\mathsf{Opt}(\Phi) \geq n/2.$$

For the sake of simpler formalism it is also convenient to assume that if a variable occurs in three clauses, then two of those literals are positive and one is negative. (In the opposite case just switch between positive and

[4] Note that a clique cover may use not only triangles and vertices but also edges, moreover there is no guarantee that an optimal solution occurs where the number of triangles is maximum.

negative for this variable in the clauses and also between true and false in its truth assignment.)

From every instance Φ satisfying the restrictions above, we construct a graph G_Φ on $8n + m$ vertices, which will be an instance of the problem in *(iii)*; cf. Figure 2. Its m *clause-vertices* will simply be denoted by $c_1, \ldots, c_m$.

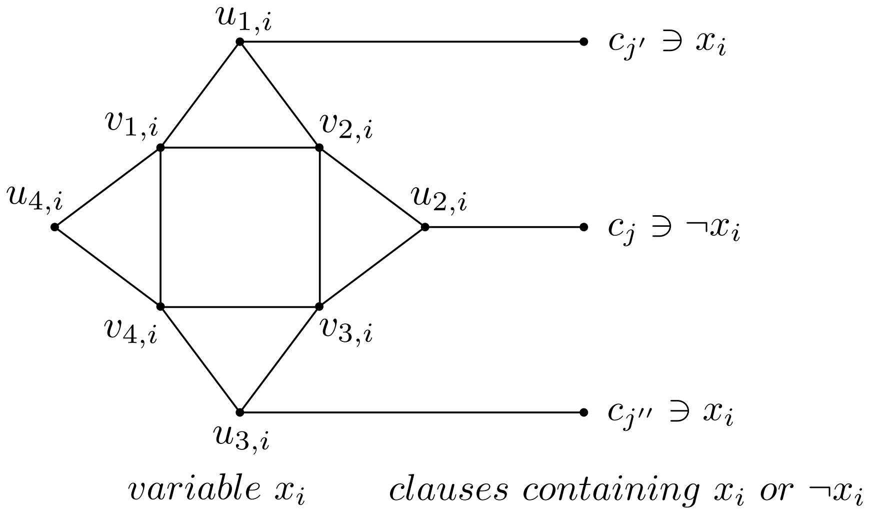

Figure 2: Variable gadget for x_i, and its adjacencies to clause vertices

The *variable-gadgets* have eight vertices each: the gadget belonging to x_i has vertex set $\{u_{1,i}, u_{2,i}, u_{3,i}, u_{4,i}, v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}\}$. The central part of the gadget is a 4-cycle $C(i) = v_{1,i}v_{2,i}v_{3,i}v_{4,i}$, its vertices appear along the cycle in this order; and the other four vertices are the third vertices of four triangles:

$$T_{1,i} = u_{1,i}v_{1,i}v_{2,i}\,, \quad T_{2,i} = u_{2,i}v_{2,i}v_{3,i}\,, \quad T_{3,i} = u_{3,i}v_{3,i}v_{4,i}\,, \quad T_{4,i} = u_{4,i}v_{4,i}v_{1,i}\,.$$

If c_j is the (unique) clause where $\neg x_i$ appears as a negative literal, then there is an edge $c_j u_{2,i}$; and if $c_{j'}, c_{j''}$ are the two clauses where x_i appears as a positive literal (say, $j' < j''$), then there are two edges $c_{j'}u_{1,i}$ and $c_{j''}u_{3,i}$. Should x_i as a positive literal occur in just one clause $c_{j'}$, only $c_{j'}$ will be adjacent with $u_{1,i}$, hence $u_{3,i}$ has then degree 2 in G_Φ. One can construct G_Φ from Φ in linear time. Moreover, we clearly have

$$\mathsf{Opt}(G_\Phi) \leq 4n + m \leq 10 \cdot \mathsf{Opt}(\Phi)$$

(where $10 \cdot \mathsf{Opt}(\Phi)$ is a very rough upper bound, but the actual coefficient of $\mathsf{Opt}(\Phi)$ is irrelevant with respect to L-reduction).

<u>*Claim.*</u> For any clique cover $\mathcal{K}$ of G_Φ there exists a clique cover $\mathcal{K}^*$ such that $|\mathcal{K}^*| \leq |\mathcal{K}|$ and, for each index i ($1 \leq i \leq n$), either $T_{1,i}, T_{3,i} \in \mathcal{K}^*$ or $T_{2,i}, T_{4,i} \in \mathcal{K}^*$.

The proof of this claim is easy but somewhat tedious since one has to consider quite a few cases. It proceeds by making local modifications inside each variable gadget. To perform them, it suffices to concentrate on those cliques $K \in \mathcal{K}$, termed *crossing clique*, which contain vertices both inside and outside the central 4-cycle $C(i)$ of the gadget in question. Note that such cliques do not have any clause vertices, and contain precisely one neighbor $u_{\ell,i}$ of $C(i)$. We make primary distinction by inspecting the positions of the intersections

$$K \cap \{v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}\},$$

with secondary distinction by the vertex of K in $\{u_{1,i}, u_{2,i}, u_{3,i}\}$ if literals of x_i appear in precisely three clauses, or in $\{u_{1,i}, u_{2,i}\}$ if each of x_i and $\neg x_i$ appears in precisely one clause.

A systematic listing of cases is given in the Appendix. As an example for the transformation, suppose that $K = \{u_{1,i}, v_{2,i}\}$ is the unique such clique. It means that $u_{2,i}$ and $u_{3,i}$ are contained in some "external" cliques, say K^2 and K^3, which do not meet $C(i)$; but some cliques of $\mathcal{K}$ must contain $u_{4,i}$, $v_{1,i}$, $v_{3,i}$, and $v_{4,i}$. This needs at least two cliques of $\mathcal{K}$ not yet listed. We then omit those two cliques from $\mathcal{K}$, and do the following further local modifications: adjoin $v_{1,i}$ to K, which yields $T_{1,i}$; remove $u_{3,i}$ from K^3, and cover it together with $v_{3,i}$ and $v_{4,i}$ as $T_{3,i}$; and take the singleton $\{u_{4,i}\}$ as an additional clique. In this way two cliques of $\mathcal{K}$ have been removed and two others inserted, while all vertices are still covered and $\{T_{1,i}, T_{3,i}\}$ is included in the modified cover. (The clique K^2 remained unchanged.) The other situations can also be handled with similar modifications, some of them leading to $|\mathcal{K}^*| < |\mathcal{K}|$, some others keeping $|\mathcal{K}^*| = |\mathcal{K}|$ similarly to the present one.

We now define a mapping from the set of solutions on G_Φ to set of solutions on Φ, specifying a truth assignment $f_{\mathcal{K}}(x_i)$ for each clique cover $\mathcal{K}$. For $i = 1, \ldots, n$ let

$$f_{\mathcal{K}}(x_i) = \begin{cases} \textsf{true} & \text{if } \{T_{2,i}, T_{4,i}\} \subset \mathcal{K}^*, \\ \textsf{false} & \text{if } \{T_{1,i}, T_{3,i}\} \subset \mathcal{K}^*. \end{cases}$$

Observe the following fact: If an x_i is a positive literal in a clause c_j, and in the variable gadget of x_i the clique cover $\mathcal{K}^*$ uses the triangles $T_{2,i}$ and $T_{4,i}$, then the value of c_j is set true by $f_{\mathcal{K}}$. A similar relation holds for negative literals in connection with $T_{1,i}$ and $T_{3,i}$. Thus, if $f_{\mathcal{K}}(c_j) = \mathsf{false}$, then $\mathcal{K}^*$ contains $\{c_j\}$ as a separate clique (cf. Figure 3), therefore

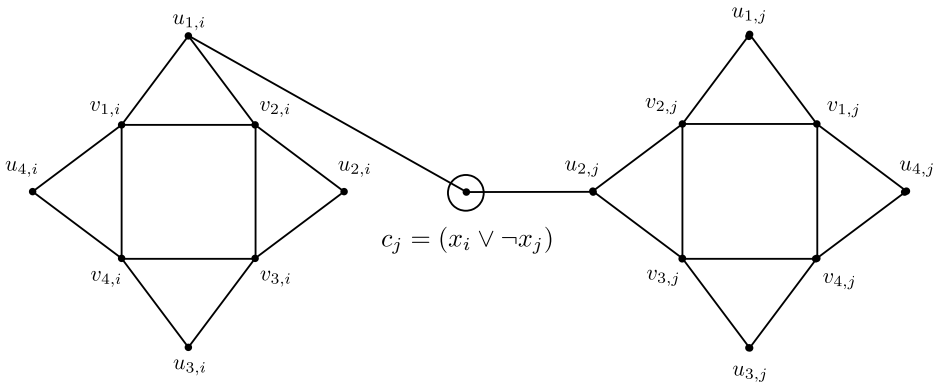

Figure 3: Non-satisfied clause yields singleton clique

$$\mathsf{val}(G_\Phi, \mathcal{K}) \geq \mathsf{val}(G_\Phi, \mathcal{K}^*) \geq 4n + m - \mathsf{val}(\Phi, f_{\mathcal{K}}). \tag{26}$$

We can define a mapping in the opposite direction, too. If f is a truth assignment in which the set of satisfied clauses is $\{c_j \mid j \in J_f\}$ (where $J_f \subseteq \{1, \ldots, m\}$), then a clique cover $\mathcal{K}_f$ can be generated by the following rules:

- if $f(x_i) = \mathsf{true}$, then put $T_{2,i}$ and $T_{4,i}$ into $\mathcal{K}_f$;
- if $f(x_i) = \mathsf{false}$, then put $T_{1,i}$ and $T_{3,i}$ into $\mathcal{K}_f$;
- for each $j \in J_f$ select an index i_j such that the literal of x_{i_j} sets c_j true;
- put the edge $c_j u_{\ell,i_j}$ into $\mathcal{K}_f$, where u_{ℓ,i_j} is the neighbor of c_j in the vertex gadget of x_{i_j};
- each vertex not covered so far is put into $\mathcal{K}_f$ as a complete subgraph of order 1.

Since the set

$$\{u_{\ell,i} \mid 1 \leq i \leq n,\ 1 \leq \ell \leq 4\}$$

is independent, every clique cover of the subgraph induced by the variable gadgets contains at least $4n$ complete subgraphs. The cover $\mathcal{K}_f$ uses further complete subgraphs only for those clauses which are not satisfied by f. Applying this to an optimal truth assignment, we obtain:

$$\mathsf{Opt}(G_\Phi) \leq 4n + m - \mathsf{Opt}(\Phi). \tag{27}$$

The combination of the inequalities (26) and (27) yields

$$\mathsf{Opt}(\Phi) + \mathsf{Opt}(G_\Phi) \leq 4n + m \leq \mathsf{val}(\Phi, f_\mathcal{K}) + \mathsf{val}(G_\Phi, \mathcal{K}),$$

thus the inequality

$$|\mathsf{Opt}(\Phi) - \mathsf{val}(\Phi, f_\mathcal{K})| \leq |\mathsf{Opt}(G_\Phi) - \mathsf{val}(G_\Phi, \mathcal{K})|$$

is valid for every solution $\mathcal{K}$ on G_Φ. It implies that we have an L-reduction, and consequently the problems listed in (i)–(iii) are APX-hard. □

Theorem 14. *The RAR-problem is* APX*-complete, even when it is restricted to the following type of instances: there are only three machines ($m = 3$), all jobs have unit time ($p_j = 1$ for all $1 \leq j \leq n$), any job can be processed on any machine ($\mathcal{M}(j) = \{M_1, M_2, M_3\}$ for all $1 \leq j \leq n$), and each resource is required only for two jobs ($|\Lambda_k| = 2$ for all $1 \leq k \leq \mu$).*

Proof. We apply Theorem 13(i), constructing a reduction from the Chromatic Number problem. Let G be a graph in which no four vertices are mutually nonadjacent. Let the vertices $v_1, \ldots, v_n$ of G represent unit-time jobs $1, \ldots, n$, and let each edge $v_i v_j \in E(G)$ mean that there is a resource $R(i, j)$ required for precisely the jobs i and j. If v_i and v_j are nonadjacent, then there is no such common resource for them. Suppose further that any of the three machines M_1, M_2, M_3 can process any of the jobs $1, \ldots, n$. Then a subset of jobs can be scheduled within a unit time slot if and only if the corresponding vertices are mutually nonadjacent. Thus, the optimum makespan for the RAR-problem instance is equal to the chromatic number of G. Consequently, the problem is APX-hard. Since membership in APX is clear (as noted above), the theorem follows. □

Corollary 15. *The SWC-problem is* APX*-hard on three machines with unit-time jobs.*

It is worth comparing this result with the complexity of SWC on two machines, as proved in [17]: it is solvable optimally in polynomial time if the processing times satisfy $p_j \in \{1, 2\}$ for all j, 4/3-approximable if $p_j \in \{1, 2, 3\}$ (the status of APX-hardness is not settled for this case), and is APX-complete if $p_j \in \{1, 2, 3, 4\}$.

5 Conclusions and Open Questions

We have studied a scheduling problem in which the jobs can only be processed on specified subsets of the machines, moreover they require simultaneous availability of renewable resources. We achieved results both in the general setting and in cases where jobs are assumed to have unit processing time and/or they require a limited number of resources. In the next three tables we summarize our results.

If $m = 1$, the RAR-problem is of no sense, as all jobs must be processed on a single machine. So we start the number of machines with $m = 2$. We abbreviate by "con" that the number m of machines or the degree B of the problem is a fixed constant, and we denote by "arb" if m or B can be arbitrarily large, being part of the input. In parentheses we indicate the corresponding statement (T1 as Theorem 1, T2 as Theorem 2, etc., and similarly, C stands for Corollary). Writing "OPT" means that the optimum can exactly be determined by an algorithm with polynomial running time, while "???" means that it is open whether the problem is APX-hard, or admits a PTAS, or can even be solved optimally by a polynomial-time algorithm.

Table 1 contains complexity results for the special case where the jobs have unit time. Here the complexity status of the RAR-problem with constant B and more than two machines is still open.

	m, the number of machines		
	$m = 2$	$m =$ con, $m \geq 3$	arb
$B = 1$	OPT (T1)	OPT (T2)	OPT (T2)
$B =$ con	OPT (T1)	???	???
$B =$ arb	OPT (T1)	APX-complete (T14)	$\Omega(n^{1-\varepsilon})$ ([17], T11)

Table 1: Complexity for unit-time jobs

Table 2 shows that unrestricted processing times make the complexity of the problem much higher for most combinations of m and B. If m or B is not constant, then the RAR-problem is proved to be APX-hard; if both m and B are constants, we do not know whether a PTAS can be designed.

	m, the number of machines		
	$m = 2$	$m =$ con, $m \geq 3$	arb
$B = 1$	NP-hard	PTAS (T10)	APX-complete (T14)
$B =$ con	NP-hard	NP-hard	APX-complete (T14)
$B =$ arb	APX-complete ([17])	APX-complete (T14)	$\Omega(n^{1-\varepsilon})$ ([17], T11)

Table 2: Complexity for arbitrary processing times

Finally, we summarize our approximation bounds in Table 3.

	m, the number of machines		
	$m = 2$	$m = con, m \geq 3$	arb
$B = 1$, arb. proc. times	PTAS (T10)	PTAS (T10)	$3 - \frac{1}{m}$ (T5)
unit times	1 (T1)	1+B (C7)	1+B (C7)
arb. proc. times	$1 + \varepsilon + B$ (C8)	$1 + \varepsilon + B$ (C8)	$2 - \frac{1}{m} + B$ (T5)

Table 3: Approximation bounds

5.1 Open Problems and Topics for Further Research

Below we list several problems which remain open or are interesting topics for future research.

- The complexity status of the RAR-problem with unit time jobs and constant degree B is still open. It is not known whether the problem is NP-hard or whether there is an exact algorithm with polynomial running time.

- For the RAR-problem with unit time jobs and a constant number of machines Corollary 7 gives a worst-case bound of $1 + B$. This is very close to the bound of $1 + B + \varepsilon$ which we get by Corollary 8 for the RAR-problem with arbitrary processing times and a constant number of machines. It seems plausible that for unit time jobs algorithms with much better bounds exist. Even the existence of an optimal algorithm cannot be excluded.

- Our algorithms are usually split into two phases. In the first phase, jobs are assigned to possible machines. In the second phase, jobs are assigned to time slots. It would be interesting to construct better algorithms by using a combined approach.

- Section 3 contains a PTAS for $B = 1$ and m fixed. Can this approach be extended to constant B or at least $B = 2$?

- Can a better approximation be given under some special assumptions on the processing times? For example, assume that only two values of processing times occur, like in the paper of Chakrabarty et al. [12]: each job is either heavy ($p_j = 1$) or light ($p_j = \varepsilon$, for some parameter $\varepsilon > 0$). Or, let the processing times be integers from a given range $\{1, 2, \ldots, p\}$.

- How is the non-approximability hardness getting worse and worse as m grows? For unit-time jobs, $m = 3$ and fixed B, the lower bound on the multiplier is a constant slightly larger than 1; and for large m we have a lower bound $O(n^{1-\varepsilon})$. How does this transition happen in detail as m grows?

- How does it make the problem harder if we have time-windows for the resources?

References

[1] R. K. Ahuja, J. B. Orlin, C. Stein and R. E. Tarjan, Improved algorithms for bipartite network flow, SIAM J. Comput., 23, 906–933, 1994.

[2] N. Alon, A simple algorithm for edge-coloring bipartite multigraphs, Inform. Process. Lett., 85 (6), 301–302, 2003.

[3] B. S. Baker and E. G. Coffman, Mutual exclusion scheduling, Theoret. Comput. Sci, 162 (2), 225–243, 1996.

[4] C. Berge, Graphs, North-Holland, Amsterdam, 1985.

[5] P. Berman and M. Karpinski, On some tighter inapproximability results (Extended abstract), Proc. ICALP'99, LNCS 1644, Springer-Verlag, 200–209, 1999.

[6] J. Blazewicz, Deadline scheduling of tasks with ready times and resource constraints, Inform. Process. Lett., 8 (2), 60–63, 1979.

[7] J. Blazewicz, J. Barcelo, W. Kubiak and H. Rock, Scheduling tasks on two processors with deadlines and additional resources, Europ. J. Oper. Res., 26, 364–370, 1986.

[8] J. Blazewicz, W. Cellary, R. Slowinski and J. Weglarz, Scheduling under resource constraints—deterministic models, Ann. Oper. Res., 7, 359 pp., 1986.

[9] J. Blazewicz, W. Kubiak, H. Rock and J. Szwarcfiter, Minimizing mean flow-time with parallel processors and resource constraints, Acta Inform., 24, 513–524, 1987.

[10] J. Blazewicz, J. K. Lenstra and A. H. G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, Discrete Appl. Math., 5 (1), 11–24, 1983.

[11] A. Caprara and R. Rizzi, Packing triangles in bounded degree graphs, Inform. Process. Lett., 84, 175–180, 2002.

[12] D. Chakrabarty, S. Khanna and S. Li, On $(1, \varepsilon)$-restricted assignment makespan minimization, Proc. Annual ACM-SIAM Symposium on Discrete Algorithms, 1087–1101, 2015.

[13] G. Dosa and Zs. Tuza, Multiprofessor Scheduling, Discr. Appl. Math., http://dx.doi.org/10.1016/j.dam.2016.01.035 (in print)

[14] T. Ebenlendr, M. Krčál and J. Sgall, Graph balancing: a special case of scheduling unrelated machines, Algorithmica, 68 (1), 62–80, 2014.

[15] E. B. Edis, C. Oguz and I. Ozkarahan, Parallel machine scheduling with additional resources: Notation, classification, models and solution methods, Europ. J. Oper. Res., 230 (3), 449–463, 2013.

[16] J. Edmonds, Paths, trees, and flowers, Canad. J. Math., 17, 449–467, 1965.

[17] G. Even, M. M. Halldórsson, L. Kaplan and D. Ron, Scheduling with conflicts: online and offline algorithms, J. Sched., 12, 199–224, 2009.

[18] M. R. Garey and D. S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM J. Comput., 4, 397–411, 1975.

[19] E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, J. ACM, 23, 317–327, 1976.

[20] H. Kellerer and V. A. Strusevich, Scheduling parallel dedicated machines under a single non-shared resource, Europ. J. Oper. Res., 147, 345–364, 2003.

[21] H. Kellerer and V. A. Strusevich, Scheduling problems for parallel dedicated machines under multiple resource constraints, Discr. Appl. Math., 133, 45–68, 2004.

[22] H. Kellerer and V. A. Strusevisch, Scheduling parallel dedicated machines with the speeding-up resource, Naval Res. Log., 55 (5), 377–389, 2008.

[23] E. L. Lawler and J. Labetoulle, On preemptive scheduling of unrelated parallel processors by linear programming, J. Assoc. Comput. Mach. 25, 612.-619, 1978.

[24] J. K. Lenstra, D. Shmoys and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Math. Progr., 46, 259–271, 1990.

[25] C.-L. Li, Scheduling to minimize the total resource consumption with a constraint on the sum of completion times, Europ. J. Oper. Res., 80, 381–388, 1995.

[26] G. Muratore, U. M. Schwarz and G. J. Woeginger, Parallel machine scheduling with nested job assignment restrictions, Oper. Res. Lett., 38, 47–50, 2010.

[27] C. H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, J. Comput. System Sci., 43, 425–440, 1991.

[28] M. Pinedo, Scheduling: Theory, Algorithms and Systems, 5th ed., Springer, New York, 2016.

[29] E. V. Shchepin and N. Vakhania, An optimal rounding procedure gives a better approximation for scheduling unrelated machines, Oper. Res. Lett., 33, 127–133, 2005.

[30] J. A. Ventura and D. Kim, Parallel machine scheduling with earliness–tardiness penalties and additional resource constraints, Computers & Oper. Res., 30 (13), 1945–1958, 2003.

[31] D. Zuckerman, Linear degree extractors and the inapproximability of Max Clique and Chromatic Number, Theory Comput., 3, 103–128, 2007.

Appendix — Cases for the Claim on page 25

The verification of the Claim in the proof of Theorem 13 can be carried out by a systematic scanning of cases. This may be organized in several ways; below we list the cases via one possible approach.

The transformation from $\mathcal{K}$ to $\mathcal{K}^*$ is done inside each variable gadget independently of the others. We recall that each variable x_i occurs as a negative literal $\neg x_i$ in precisely one clause c_j, represented by the edge $c_j u_{2,i}$ of G_Φ. Moreover, the positive literal x_i occurs in either just one clause $c_{j'}$ or two clauses $c_{j'}$ and $c_{j''}$, represented by the single edge $c_{j'}u_{1,i}$ or by the two edges $c_{j'}u_{1,i}$ and $c_{j''}u_{3,i}$. (The latter situation is exhibited in Figure 2.) In the parts (1) and (2) below, we list the possible subcases for the single and for the double occurrences of literal x_i, respectively. The position of crossing clique(s) is explicitly given after the '$\rightarrow$' sign in each subcase.

We assume in all subcases that the complete subgraphs (edges and triangles) of the initial clique cover are mutually vertex-disjoint. In case if the positive variable x_i occurs in two clauses of Φ, we say that an edge or a triangle in the cover of the corresponding variable gadget is *crossing* if it has at least one vertex in each of the sets $\{u_{1,i}, u_{2,i}, u_{3,i}\}$ and $\{v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}\}$. More restrictively if x_i has only one positive (and one negative) appearance in Φ, then a crossing clique has one vertex in $\{u_{1,i}, u_{2,i}\}$ and one or two in $\{v_{1,i}, v_{2,i}, v_{3,i}\}$. (Hence, cliques are not considered to be crossing if they do not contain any vertices adjacent to a clause vertex; this is the reason for the absence of $u_{3,i}$ and its neighbor $v_{4,i}$ in the definition if x_i does not have a second positive occurrence in Φ.)

To ensure a transparent systematic listing, each case description begins with a number, or with a combination of numbers, expressing how many vertices the crossing cliques have in $\{v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}\}$. For instance, '1 + 2' with its two terms means that there are two crossing cliques in the clause gadget under investigation, one of them (a crossing edge, represented with '1') has one vertex in $\{v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}\}$, and the other one (a crossing triangle, belonging to '2') has two of the v-vertices.

There are situations where already those numbers determine the positions of the crossing cliques up to symmetry. This is the case e.g. with '$1+2$' when we are in (1). Indeed, then the corresponding crossing edge and crossing triangle involved in the clique cover together entirely contain $\{u_{1,i}, u_{2,i}\} \cup \{v_{1,i}, v_{2,i}, v_{3,i}\}$. Thus, the cliques are either $\{u_{1,i}, v_{1,i}, v_{2,i}\}$ and $\{u_{2,i}, v_{3,i}\}$ or $\{u_{1,i}, v_{1,i}\}$ and $\{u_{2,i}, v_{2,i}, v_{3,i}\}$, these two possibilities being symmetric to each other. Such symmetric variants, which therefore need not be checked separately, are indicated in brackets in the list.

In other situations the positions of crossing cliques are not uniquely determined by the numbers alone. Note that crossing cliques never contain the edge $v_{1,i}v_{4,i}$, therefore we may view the v-part of them as a subset (or subgraph) of the path $v_{1,i}v_{2,i}v_{3,i}$ in (1) or $v_{1,i}v_{2,i}v_{3,i}v_{4,i}$ in (2). Those paths have two end vertices and one or two middle vertices, which are referred to as 'end' and 'middle' in the list, respectively. In most of the subcases, this is our main way to distinguish between the possible positions of crossing cliques. Differently from this, in subcase '$1+1$' of (2) we make primary distinction according to the distance of the corresponding two vertices in $\{v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}\}$.

We simplify notation and omit the second subscript 'i' from $u_{\ell,i}$ and $v_{\ell,i}$ for $1 \le \ell \le 4$. It is a matter of routine to check that, in each case, the clique cover can be modified to one which does not contain more cliques than the original cover, and which includes either $T_{1,i}$ and $T_{3,i}$ or $T_{2,i}$ and $T_{4,i}$.

As an illustration, consider the case '$1+2$' with 'middle pair' in (2). It assumes the presence of $K^1 = \{u_1, v_1\}$ and $K^2 = \{u_2, v_2, v_3\}$ in the initial clique cover. Such a cover can be modified to one with the required property by replacing its clique containing u_4 with $\{u_4, v_1, v_4\}$. This transformation reduces K^1 to the singleton $\{u_1\}$ (and if the initial cover was not locally optimal inside the variable gadget, then eliminates the singleton clique $\{v_4\}$). Similar modifications can be done in all subcases; details are left to the reader.

(1) — only u_1 and u_2 are adjacent to clause vertices $\Rightarrow$ crossing cliques meet $C(i)$ in a subset of $\{v_1, v_2, v_3\}$

possible cardinalities of the crossing clique intersections with $\{v_1, v_2, v_3\}$:

- 0 $\rightarrow$ no crossing cliques
- 1 $\rightarrow$ two subcases
    - end (v_1 [or v_3]) $\rightarrow K = \{u_1, v_1\}$
    - middle (v_2) $\rightarrow K = \{u_1, v_2\}$ [or $K = \{u_2, v_2\}$]
- 2 (v_1v_2 [or v_2v_3]) $\rightarrow K = \{u_1, v_1, v_2\}$
- $1 + 1 \rightarrow$ two subcases
    - two ends (v_1 and v_3) $\rightarrow K^1 = \{u_1, v_1\}$, $K^2 = \{u_2, v_3\}$
    - includes middle (v_1, v_2 [or v_2, v_3]) $\rightarrow K^1 = \{u_1, v_1\}$, $K^2 = \{u_2, v_2\}$
- $1 + 2$ (v_1v_2 and v_3 [or v_1 and v_2v_3]) $\rightarrow K^1 = \{u_1, v_1, v_2\}$, $K^2 = \{u_2, v_3\}$

(2) — all the three u_1, u_2, u_3 are adjacent to clause vertices $\Rightarrow$ crossing cliques meet $C(i)$ in a subset of $\{v_1, v_2, v_3, v_4\}$, their edge set does not contain edge v_1v_4

possible cardinalities of the clique intersections with $\{v_1, v_2, v_3, v_4\}$:

- 0 $\rightarrow$ no crossing cliques
- 1 $\rightarrow$ two subcases
    - end (v_1 [or v_4]) $\rightarrow K = \{u_1, v_1\}$
    - middle (v_2 [or v_3]) $\rightarrow$ two subcases
        - $K = \{u_1, v_2\}$
        - $K = \{u_2, v_2\}$
- 2 $\rightarrow$ two subcases
    - end (v_1v_2 [or v_3v_4]) $\rightarrow K = \{u_1, v_1, v_2\}$
    - middle (v_2v_3) $\rightarrow K = \{u_2, v_2, v_3\}$
- $1 + 1 \rightarrow$ three subcases
    - neighbors $\rightarrow$ two subcases
        - v_1 and v_2 [or v_3 and v_4] $\rightarrow K^1 = \{u_1, v_1\}$, $K^2 = \{u_2, v_2\}$

        - v_2 and v_3 $\to$ two subcases
            - $K^1 = \{u_1, v_2\}$, $K^2 = \{u_2, v_3\}$
            - $K^1 = \{u_1, v_2\}$, $K^2 = \{u_3, v_3\}$
    - distance 2 (v_1 and v_3 [or v_2 and v_4]) $\to$ two subcases
        - $K^1 = \{u_1, v_1\}$, $K^2 = \{u_2, v_3\}$
        - $K^1 = \{u_1, v_1\}$, $K^2 = \{u_3, v_3\}$
    - two ends (v_1 and v_4) $\to$ $K^1 = \{u_1, v_1\}$, $K^2 = \{u_3, v_4\}$
- $1 + 2$ $\to$ two subcases
    - middle pair (v_2v_3 with v_1 [or v_4]) $\to$ $K^1 = \{u_1, v_1\}$, $K^2 = \{u_2, v_2, v_3\}$
    - end pair (v_1v_2 [or v_3v_4]) $\to$ two subcases
        - end single (v_4) $\to$ $K^1 = \{u_1, v_1, v_2\}$, $K^2 = \{u_3, v_4\}$
        - middle single (v_3) $\to$ two subcases
            - $K^1 = \{u_1, v_1, v_2\}$, $K^2 = \{u_2, v_3\}$
            - $K^1 = \{u_1, v_1, v_2\}$, $K^2 = \{u_3, v_3\}$
- $2 + 2$ (v_1v_2 and v_3v_4) $\to$ $K^1 = \{u_1, v_1, v_2\}$, $K^2 = \{u_3, v_3, v_4\}$
- $1 + 1 + 1$ $\to$ two subcases
    - missing end (v_1, v_2, v_3 [or v_2, v_3, v_4]) $\to$ $K^1 = \{u_1, v_1\}$ $K^2 = \{u_2, v_2\}$, $K^3 = \{u_3, v_3\}$
    - missing middle (v_1, v_2, v_4 [or v_1, v_3, v_4]) $\to$ $K^1 = \{u_1, v_1\}$ $K^2 = \{u_2, v_2\}$, $K^3 = \{u_3, v_4\}$
- $1 + 1 + 2$ $\to$ two subcases
    - end pair (v_1v_2 with v_3 and v_4 [or v_3v_4 with v_1 and v_2]) $\to$ $K^1 = \{u_1, v_1, v_2\}$ $K^2 = \{u_2, v_3\}$, $K^3 = \{u_3, v_4\}$
    - middle pair (v_2v_3 with v_1 and v_4) $\to$ $K^1 = \{u_1, v_1\}$ $K^2 = \{u_2, v_2, v_3\}$, $K^3 = \{u_3, v_4\}$