INSTITUTE OF
ACTUARIAL SCIENCE
&
ECONOMETRICS

REPORT AE 6/2004

On the complexity of Adjacent Resource Scheduling

C.W. Duin
E. van der Sluis

University of Amsterdam

ABSTRACT

We study the problem of scheduling resource(s) for jobs in an adjacent manner (ARS). The problem relates to fixed interval scheduling on the one hand, and to the problem of two-dimensional strip packing on the other hand. Further, there is a close relation with multiprocessor scheduling. A distinguishing characteristic is the constraint of resource-adjacency.

As an application of ARS, consider an airport where passengers check in for their flight, joining lines before one or more desks; at the desk the luggage is checked and so forth. To smooth these operations the airport maintains a clear order in the waiting lines: a number *n(f)* of *adjacent* desks is to be assigned exclusively during a fixed time-interval *I(f)* to flight *f*. For each flight in a given planning horizon of discrete time periods, one seeks a feasible assignment to adjacent desks and the objective is to minimize the total number of involved desks.

The paper explores two problem variants and relates them to other scheduling problems. The basic, rectangular version of ARS is a special case of multiprocessor scheduling. The other problem is more general and it does not fit into any existing scheduling model.

After presenting an integer linear program for ARS, we discuss the complexity of both problems, as well as of special cases. The decision version of the rectangular problem remains strongly NP-complete. The complexity of the other problem is already strongly NP-complete for two time periods. The paper also determines a number of cases that are solvable in polynomial time.

KEYWORDS: combinatorial optimization, multiprocessor scheduling, strip packing, integer programming

1. Introduction

Deterministic problems involving the scheduling or sequencing of jobs on machines form a well-studied group of problems in the literature; for an overview see, e.g., Lawler et al. (1993), Pinedo (2001) or Blazewicz et al. (2001). The problem field is nowadays highly categorized as to machine-environment, processing characteristics of jobs and objectives of scheduling. For many problems the complexity status and -if NP-hard- the degree of approximability is known, see Chen et al. (1998).

However, the problem of adjacent resource scheduling (ARS) has been overlooked more or less. This problem considers the maximum resource level as needed for a number of jobs. It has interesting applications when jobs are to use a number of resource units during a (given) time-interval *in an adjacent manner*. E.g., consider the scheduling of desks for checking-in passengers at the airport. Here jobs are flights, each requiring an adjacent number of desks during a specified period. This application, reviewed in more detail in section 2, is due to Chun (1996); he also describes more-dimensional variants of the problem. The complexity of ARS was not discussed.

ARS is a sequencing problem within the category of a fixed-interval scheduling. On the other hand, there is an even closer relation between ARS and the field of multiprocessor scheduling; for an overview of this field, see Drozdowski (1996). Finally, ARS relates to the problem fields of cutting stock or packing bins in two-dimensions, see, e.g., Dyckhoff (1990) and Lodi et al. (2002).

This paper analyses the complexity of two variants of ARS: a basic version, named ARS-R, considers 'rectangular' jobs and a more general version, ARS-V, considers jobs with resource needs that vary over time. From resource-constrained project scheduling we adapt an integer linear programming model to solve ARS-V.

Problem ARS-V does not fit into existing scheduling models but ARS-R is a special case of a known multiprocessor scheduling problem with dedicated processors, namely $P \mid fix_j \mid C_{max}$, see Blazewicz et al. (1986). In view of its applications a separate complexity analysis of the special case, ARS-R, is mandatory. We will see that ARS-R remains strongly NP-complete, but under restrictive circumstances its complexity status is different from $P \mid fix_j \mid C_{max}$, e.g., when resource needs are constant, or, when the time horizon consists of at most three periods. A polynomial time approximation scheme (PTAS) exist for the problem ARS-R[T_0], which considers only the instances of ARS-R with the number of time periods T fixed at some constant T_0. This problem inherits the PTAS as developed for $Pm \mid fix_j \mid C_{max}$ in Amoera et al. (1997).

In section 3 we first show that ARS-R remains NP-complete. Next, we show that ARS-R is strongly NP-complete. Further, we treat the complexity of special cases of ARS-R and ARS-V, e.g., if ARS-V involves only two time periods, then it remains strongly NP-complete.

2. The problem as a scheduling problem

Before defining problem ARS mathematically, we describe some motivating applications.

Most commercial airports let passengers check-in for their flight by joining queues before one or more desks; at this desk luggage is labeled and boarding passes are provided. To direct the passengers smoothly through these operations, the airport and airline companies wish to maintain a clear order in the waiting lines. Therefore adjacent desks are temporarily labeled as to devote their time during certain time-slots exclusively to certain flights. Given the departure time, the expected number of passengers and the nature of each flight, a desired number of desks is known at forehand: for each flight, for every period in the time slot of checking-in for that flight, a given number of counters must be available in adjacent manner. Within this very restrictive context, assuming time- as well as desk-adjacency, a still interesting scheduling problem remains to be solved: what is the minimal number of desks required in a feasible schedule?

Warehouse Management is another field of application. Often clients need temporary storage renting a number of positions within the warehouse. For reasons of convenience and cost-efficiency the commodities of each client are to be put in adjacent positions. Note that the storage-capacity as rented by a client can vary over time due to seasonal influences.

A large-scale application, known as the berthing problem, is the assignment of quay space when ships of different length moor during specified time periods. An application on micro-scale would be the assignment of mainframe computer memory or hard disk memory, that is, if a contiguous allocation of such memory over the different jobs were required. With a requirement of adjacency that is less strict, the reservation of hotel rooms can be seen an application of ARS: when guests within different groups (jobs) desire adjacent rooms.

2.1 Problem variants

Adding mathematical notation and using more general terms, with jobs instead of flights and resource units instead of desks, we can state the central problem as follows:

ARS-R A 'bank' of adjacent resource units $r=1,2,\ldots,R$ is to handle jobs $j=1,2,\ldots,J$ in a planning horizon with periods $t=1,2,\ldots,T$. Each resource unit can be assigned to at most one job per

period. During an interval $I(j)=[a(j),b(j)]$ (of $b(j)-a(j)+1$ time periods in $[1,T]$), each job j is to be assigned an invariant interval of $n(j)$ resource units within $[1,R]$. Is this possible?

By describing the involved subsets of integers as intervals we emphasize the aspects of adjacency in this problem. The second R in the acronym ARS-R refers to the 'rectangular' variant of the problem, where each j has a constant need of $n(j)$ units in periods $t \in I(j)$. Figure 1 provides for a problem instance two schedules, one that does not accomplish $R=9$ in a feasible manner and one that does. Graphically, resource-units constitute rows and time-periods columns. An instance of ARS-R can be summarized by three vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{n} \in \mathbb{N}^J$. In the example $\boldsymbol{a}$ =(1,2,3,5,6,6), $\boldsymbol{b}$ = (5,5,9,8,9,9) and $\boldsymbol{n}$ = (2,2,2,2,2,3). In Figure 1 a simple greedy heuristic named 'First-Fit' gets stuck.

Algorithm 'FIRST-FIT' Schedule the jobs j in the order of release time $a(j)$ at the first available resource row where it fits.

Infeasible

r \ t	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	5	5	5	5
2	1	1	1	1	1	5	5	5	5
3		2	2	2	2	6	6	6	6
4		2	2	2	2	6	6	6	6
5			3	3	3	3	3	3	3
6			3	3	3	3	3	3	3
7					4	4	4	4	
8					4	4	4	4	
9						6	6	6	6

Feasible

r \ t	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	6	6	6	6
2	1	1	1	1	1	6	6	6	6
3		2	2	2	2	6	6	6	6
4		2	2	2	2	5	5	5	5
5						5	5	5	5
6					4	4	4	4	
7					4	4	4	4	
8			3	3	3	3	3	3	3
9			3	3	3	3	3	3	3

Figure 1 Schedules for an instance of ARS-R. Left: Job 6 is assigned to non-adjacent rows

The mentioned applications may require a more flexible model. In ARS-R one rigidly assigns, during time interval $I(j)$, job j to a constant number $n(j)$ of resource-units. This may not adequately model the situation where a lower service is adequate during, e.g., early and/or final periods of $I(j)$. The generalization ARS-V of the problem takes care of this.

ARS-V Resource needs vary: Job j is to be assigned in period $t \in I(j)$ a resource-interval of $n(j,t)$ (>0) units in $[1,R]$ such that time-adjacent resource-intervals overlap maximally.

The constraint of maximum overlap, as imposed in ARS-V, stipulates that jobs must re-utilize a number of $min\{n(j,t), n(j,t-1)\}$ units in any (next) period $t \in (a(j),b(j)]$. Applications typically desire

this constraint to minimize the costs associated with change. E.g., in the practice of flights and desks the constraint minimizes for the data *n(j,t)* the number of times that a desk is either opened of closed. The numbers *n(j,t)* can be restricted to positive values only, since requirements *n(j,t)*=0 and *n(j,t*+1)>0, are just as well accommodated by a new job *j'* with *a(j')*=*t*+1.

a) job data n(j,t)

j \ t	1	2	3	4
1	4	1	0	0
2	2	1	2	0
3	0	4	2	1
4	0	0	1	2
5	0	0	0	3

b) feasible solution

r \ t	1	2	3	4
1	1	1		4
2	1	3	4	4
3	1	3	3	3
4	1	3	3	5
5	2	3	2	5
6	2	2	2	5

c) infeasible solution

r \ t	1	2	3	4
1	1	3	3	3
2	1	3		4
3	1	3	4	4
4	1	3	3	5
5	2	1	2	5
6	2	2	2	5

Figure 2 An ARS-V instance (a), with feasible solution (b) and infeasible solution (c)

An example of ARS-V is given in Figure 2a; a feasible (and optimal) solution is presented in Figure 2b. Figure 2c presents an infeasible solution: job 1 has not a maximal overlap at time *t*=2 and job 3 does not satisfy the desk-adjacency condition at time *t*=3.

We introduced ARS-R and ARS-V as feasibility problems. In the optimization version of these problems one seeks an optimal schedule, feasible for a minimal $R=R^*$. For any period *t*, let the set $A(t)=\{j \mid t \in I(j)\}$ comprise the jobs active in period *t*. Clearly a lower bound for R^* is the highest sum of simultaneous demands: $Rlow = \max_t \sum_{j \in A(t)} n(j, t)$. In Figure 1 and 2 we have $R^* = Rlow$.

An instance of ARS-V can contain up to four different types of jobs according to the changes of *n(j,t)* in relation to *n(j,t*-1*)* for periods *t, t*-1$\in I(j)$. All four types are present in the example of Figure 2. We have one constant, rectangular job (*j*=5); two 'decreasing' jobs (*j*=1,3) with $n(j,t) \leq n(j,t-1)$ for $t \in (a(j),b(j)]$; one increasing job (*j*=4) and one general job (*j*=2). Problem ARS-R is the special case of ARS-V where all jobs are constant. By definition a constant job is increasing as well as decreasing.

Complexity aspects of ARS-R may be different than of ARS-V. Likewise one may consider, as an in-between problem, ARS*mono*: the special case of ARS-V with in its instances either only decreasing jobs or only increasing jobs (by an argument of mirroring, an instance with decreasing jobs only, is equivalent to an instance with only increasing jobs). Easier problem variants may also arise, when restricting the problem parameters *T* or *Rlow* to some constant.

2.2 Related problems

We discuss here relations that problem ARS has with problems known from literature, for a more detailed description of the scheduling problems that we will refer to, see Blazewicz et al. (2001).

Without the restriction of resource-adjacency ARS-R is known as the reservation problem or the problem of fixed-interval scheduling. Here one minimizes the number of parallel machines to complete n jobs in given time intervals. E.g., the jobs are the orders at a car-rental company.

When picturing solutions of ARS-R, there is a conspicuous resemblance with *two-dimensional strip-packing* problem (2SP), from the field of two-dimensional packing, see Lodi et al. (2002).

2SP **W**ithin a strip of fixed width W, position n rectangles of size $w_i \times h_i$ (i=1,2, ..., n) without overlap and rotation (w_i parallel to W), such that the strip's height H is minimal.

The difference is that ARS-R is more restrictive. ARS-R would arise if one specifies in 2SP for each item a compulsory horizontal position, such that rectangles j may only shift vertically, between the given width-lines $w=a(j)$ and $w=b(j)$.

As a scheduling problem ARS-R bears also similarity to the resource-constrained project scheduling problem (RPSP). In this problem non-preemptive jobs j=1, 2, ..., n have processing times $p(j)$ during which they demand $r(j)$ units of a resource. One seeks a job schedule, meeting a given due-date, while using at any time at most R resource units simultaneously. The resemblance is less striking than at first glance. In RPSP there are precedence relations between jobs and one is allowed to shift the jobs within the interval of earliest starting- and latest finishing time, while ARS-R is frozen in time. More importantly, resource units are interchangeable in RPSP. If ARS-R would similarly allow the assignment of non-adjacent resource-units to jobs, then the schedule of Figure 1 (left) would be feasible with job 6 on resource rows 3, 4 and 9.

Suppose now that we swap the dimensions t and r. Then adjacent resource-units become (natural) adjacent time units and, somewhat unnaturally, jobs j are to be executed on 'machines' $a(j)$, $a(j)+1, ..., b(j)$, during $n(j)$ time periods. Furthermore, these machines must operate simultaneously.

Simultaneous operations are a main characteristic of *multiprocessor scheduling*, a research field with applications in parallel systems of manufacturing and computing. We describe two basic problems where one minimizes the makespan of n tasks (of which no two may occupy the same processor at any time). The first, named $P \mid size_j \mid C_{max}$, considers m identical processors and tasks j=1, ..., n that require a number of m_j ($\leq m$) processors during their processing time p_j. Thus in

$P \mid size_j \mid C_{max}$ the assignment of jobs to processors is part of the problem and this is also the case in the 'parallel task system', a generalization studied in Du and Leung (1989). To accommodate ARS-R, the tasks should have predetermined, dedicated processors, as distinguished in the second problem.

$P \mid fix_j \mid C_{max}$ Given is a system $M=\{M_1, M_2, ..., M_m\}$ of m different processors. Tasks $j=1,2, ..., n$ take p_j time and require a fixed set $\mu_j \subset M$ of processors. Is there a schedule of length < R?

After swapping time and place, ARS-R (but not so ARS-V) arises as special case of $P \mid fix_j \mid C_{max}$ in this way:

let each set μ_j consist of 'an interval' of consecutively numbered processors.

This has a rather nice consequence for ARS-R. Moera et al. (1997) show that a polynomial time approximation scheme (PTAS) exists for $Pm \mid fix_j \mid C_{max}$, where the number processors is fixed to a constant m. Likewise we defined in section 1 the problem ARS-R[T_0], with the number of time periods T fixed to some constant T_0. Since ARS-R[T_0] is again a special case of $Pm \mid fix_j \mid C_{max}$, with $m=T_0$, problem ARS-R[T_0] also allows a PTAS.

On the other hand as ARS-R is a rather special case of $P \mid fix_j \mid C_{max}$ it could very well be easier than $P \mid fix_j \mid C_{max}$. In section 3.4, we prove that ARS-R remains strongly NP-complete. However, some specializations of $P \mid fix_j \mid C_{max}$, as known from literature, are more difficult than the corresponding version of ARS-R.

For example consider within $P \mid fix_j \mid C_{max}$ the problem $P \mid fix_j, p_j=1, |\mu_j|=2, load=3 \mid C_{max}=3$. Here, there are only tasks of unit processing time on two dedicated processors such that each processor has a load of three tasks; one asks whether the makespan is 3. Following Kubale (1987) we can easily see that this very special problem case is already strongly NP-complete. The edge-coloring problem in a three-regular graph is (strongly) NP-complete, see Hoyler (1981). It transforms in pseudo-polynomial time to $P \mid fix_j, p_j=1, |\mu_j|=2, load=3 \mid C_{max}=3$ by defining for each vertex a unique processor and for each edge a unique job, to be processed in unit-time on the two incident 'processors'.

Below we list cases of ARS-R (to be inspected later) that are solvable in polynomial time, whereas the corresponding multiprocessor problem is not.

- $P \mid fix_j, p_j=1 \mid C_{max}$, with all processing times 1, is strongly NP-complete. ARS-R with all resource needs *n(j)* equal to 1, solves easily (section 2.4).
- $P \mid fix_j, |\mu_j|=2 \mid C_{max}$, with jobs on two dedicated processors, is strongly NP-complete. ARS-R solves in polynomial time, if all jobs require at most two periods (section 3.5)

- P | fix_j | C_{max}=3, the question whether a schedule of makespan 3 exists, is strongly NP-complete; see also Hoogeveen et al. (1994). A corresponding case of ARS-R, with *R* fixed to some constant number *N*, is solvable in polynomial time (section 3.5).
- P3 | fix_j | C_{max} is strongly NP-hard, see Blazewicz et al. (1992). ARS-R with *T*=3 is solvable in linear time using so-called normal schedules (section 2.4).

A generalization of problem P | fix_j | C_{max}, named general multiprocessor scheduling is more flexible in the sense that jobs *j* can be run on a family $F_j = \{\mu_{ji} \mid i=1,2, ..., |F_j|\}$ of alternative sets of processors. Chen and Miranda (2001) have extended the PTAS of P*m* | fix_j | C_{max} to the fixed version of this problem, named P*m* | set | C_{max}. Again, the generalization ARS-V of ARS-R does not fit into these models.

2.3 An ILP formulation

The likeness of RPSP and ARS-V makes it possible to adapt a well-known integer linear program of RPSP; see, e.g., Pinedo and Chun (1999). Model ILP for ARS-V, as given below, uses binary variables $y_{j,t,r}$, equal to 1 if job *j* occupies *r* as last row in period *t*. With $n^+(j,t)$ we denote nonnegative increases in demand, max {0, *n(j,t)-n(j,t-*1*)*}, and $n^-(j,t)$=max {0, *n(j,t-*1*)-n(j,t)*}. Further *A(t)* denotes the set of jobs that are active in period *t*.

ILP: min R^*

s.t.

$$\sum_{r=n(j,t)}^{R} r\, y_{j,t,r} \le R^* \qquad \forall j, \forall t \in [a(j), b(j)] \qquad (1)$$

$$\sum_{r=n(j,t)}^{R} y_{j,t,r} = 1 \qquad \forall j, \forall t \in [a(j), b(j)] \qquad (2)$$

$$\sum_{j \in A(t)} \sum_{s=r}^{r+n(j,t)-1} y_{j,t,s} \le 1 \qquad \forall r, \forall t \in [1, T] \qquad (3)$$

$$\sum_{r=1}^{R} r\, y_{j,t,r} - \sum_{r=1}^{R} r\, y_{j,t-1,j} \le n^+(j,t) \qquad \forall j, \forall t \in (a(j), b(j)] \qquad (4)$$

$$\sum_{r=1}^{R} r\, y_{j,t-1,r} - \sum_{r=1}^{R} r\, y_{j,t,r} \le n^-(j,t) \qquad \forall j, \forall t \in (a(j), b(j)] \qquad (5)$$

$$y_{j,t,r} \in \{0,1\} \qquad \forall j, \forall t, \forall r \qquad (6)$$

The *LHS* of (1) represents the last row occupied by job *j* in period *t*. Constraint (2) ensures that each job seeks one resource row as last row. To explain (3), note that job $j \in A(t)$ occupies at time *t* the resource row *r* if and only if $y_{j,t,s} = 1$ for some index $s \in [r, r+n(j,t)-1]$. So constraint (3) ensures that at

most one job uses row r in period t. The constraint of maximum overlap between adjacent resource-intervals is modelled relatively easy by constraints (4) and (5).

For ease of explanation we have used in the above model $O(JRT)$ binary variables and $O(JT+RT)$ constraints; actually, fewer variables suffice. Let $v(j)$ denote the number of periods t in $[a(j),b(j)]$ that the resource need of a job j changes, $n(j,t)\neq n(j,t\text{-}1)$ and let us write V for the total number of variations, $V=\sum_j v(j)$ (note that $V=J$ for instances of ARS-R). A reduced and improved model can be formulated with $O(VR)$ variables and constraints, see Al-Ibrahim et al. (2004).

2.4 Divide and conquer

This section uncovers some properties of problems ARS-R and ARS-V. We prefer to formulate them only for version ARS-R; the stated properties extend in an obvious manner to ARS-V.

The constraint of resource-adjacency is a defining characteristic of problem ARS-R. When it is relaxed such that one may use non-adjacent resource-rows, then the problem is well solvable.

Proposition 1 ARS-R without the restriction of resource-adjacency, solves easily with $R^* = Row$.

Proof Split each job j into identical jobs j_k for k=1,2, ..., $n(j)$ with $I(j_k) = I(j)$ but with unary demand $n(j_k) = 1$. Algorithm FIRST-FIT finds for each (unary) job a location on a (single) row $\leq Rlow$.□

Mind that the algorithm as adapted from the reservation problem in the proof of proposition 1 takes time $O(\sum_j n(j)) = O(JR)$. In our context this would be a pseudo-polynomial algorithm, because the input length of an instance of ARS-R has size $O(J\,logT + J\,logR)$. Using the right data structures, it is however possible to derive a strongly polynomial time algorithm for problem ARS-R without adjacency: by sorting the jobs according to their demand $n(j)$ and splitting them up parsimoniously, only where and when necessary for placement at the first available location.

Even so, for applications like the check-in process J times R is a relatively small number. Furthermore, the subtlety disappears in the next application of proposition 1.

Corollary 2 The subproblem of ARS-R with all jobs having unary demand, i.e., $n(j) = 1$ for all j, is solvable in linear time.

An opposite extreme case, where each job has a time slot of just one period, is also solvable in linear time. This property also follows from a repeated application of the following observation.

Proposition 3 Let $s > 1$ be a period such that all jobs with start $a(j) \leq s$ also have $b(j) \leq s$. Then one can decompose the instance into two smaller instances: problem 1 contains the jobs j with $I(j) \subset [1,s]$, and problem 2 the other jobs with $I(j) \subset [s+1,T]$.

After such a decomposition one joins the two sub-schedules with outcomes R_1^* and R_2^*, say, obtaining this way for the original problem $R^* = \max\{R_1^*, R_2^*\}$. Having seen this possibility of reduction of T, one may wonder under which circumstances a reduction is possible of R. Trivially one can also eliminate any job j with $|I(j)| = T$, when afterwards R^* is augmented with $n(j)$. Extending this to a rule for a reduction of R, consider a set of time-adjacent jobs, filling up the entire interval $[1,T]$ with equally sized resource-intervals. As shown later in the next section, the elimination of such jobs is not valid. (For readers acquainted with the game 'Tetris': the vanishing rule does not apply.)

One can easily obtain $R^*=Rlow$ in special cases with a small time horizon. Remember that ARS-R[T_0] is the subproblem of ARS-R containing all instances with planning horizon length $T = T_0$.

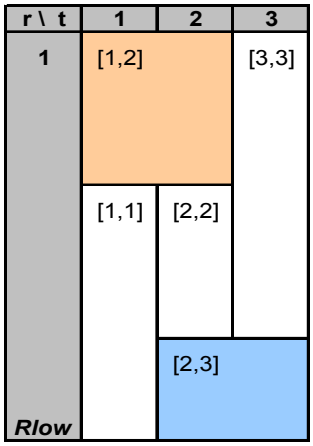

Figure 3 Solving ARS_R(3) by means of a normal schedule

Problem ARS-R[1] is trivially solvable with $R^*=Rlow$ and this is also not difficult for ARS-R[2] and ARS-R(3). One partitions the jobs in groups with equal interval $[a(j),b(j)]$ in order to schedule the jobs within a group consecutively; such schedules are called 'normal schedules'. For ARS-R[2] one achieves $R^*=Rlow$ by assigning the 'wide' jobs j with $a(j)=1$ and $b(j)=2$ to the first rows.

For ARS-R[3] one can also attain $R^*=Rlow$ in linear time, as is illustrated in Figure 3. Again, the full-length jobs with $a(j)=1$ and $b(j)=3$ are assigned to the first rows. Then according to their time periods, five types of jobs remain, we label them: there are three types of 1-period jobs, say the [1,1]-, [2,2]-, and [3,3]-jobs and there are two types of 2-period jobs, the [1,2]-jobs and [2,3]-jobs. Suppose that the jobs in the group of [1,2]-jobs occupy row $r=1$ and that the group of [2,3]-jobs takes $r=Rlow$.

Then this partial arrangement leaves a blank interval in each column for job groups [1,1], [2,2], and [3,3], to determine with [1,2]- and/or [2-3]-jobs the value of *Rlow*. Summarizing, we have:

Proposition 4 Problem ARS-R[T_0] solves in linear time for $T_0 \leq 3$ by means of normal schedules.

3. Problem complexities

With a program like *ILP* of section 2.3 many instances of ARS-R were solved, each time with R^* equal to *Rlow*. Larger problems did not finish in a time limit and for some time we wondered: Do all instances of ARS-R solve with $R^* = Rlow$? To answer this question, we design a special instance.

3.1 The instance ARS-R9(*G*)

This instance, given below as ARS-R9(*G*), consists of 9 jobs. As we will show ARS-R9(*G*) essentially has a unique solution with $R^* = Rlow$, such that there are two holes of size *G* in the same column. The number *G* is an arbitrarily chosen integer ≥2. For all periods, except for period 3, the total resource demand is $2G+3 = Rlow$.

j	1	2	3	4	5	6	7	8	9
n(j)	*G*+1	*G*+1	*G*	*G*+1	2*G*+1	2*G*+2	1	1	1
a(j)	1	1	2	2	4	5	1	2	3
b(j)	1	1	2	2	4	5	3	4	5

Figure 4a The data of ARS-R9(*G*) with *G* a given integer ≥ 2

\ t	1	2	3	4	5
1	7		7	5	6
2	1				
…					
G+1					
G+2	1				
G+3	2				
…				5	
2*G*+2		8		8	6
2*G*+3	2		9		9

r \ t	1	2	3	4	5
1	1	8		8	6
2		3		5	
…					
G+1	1	3			
G+2	7		7		
G+3	2	4			
…					
2*G*+2				5	6
2*G*+3	2	4	9		9

Figure 4b Scheduling ARS-R9(*G*) within 2*G*+3 resource rows

Proposition 5 An optimal solution of ARS-R9(*G*) has two gaps of size *G* in column 3.

Proof To obtain an optimal solution for ARS-R9(G), we need to consider only two cases for job 7: an assignment to row 1 or to row G+2. Other assignments ($2G$+3 is not discussed as it is symmetric to 1), will push R^* above *Rlow* already in column t=1, because of the indivisibility of jobs 1 and 2.

***Job 7 assigned to row* 1**: the first row available for job 1 is row 2; the first row for job 2 is row G+3. Now, other jobs that can be assigned to row 1 are jobs 5 and 6, i.e. jobs with demand only in period 4 and 5 respectively. So, both these jobs need to start at row 1. Consequently, the next assignments are job 9 to $2G$+3 and job 8 to $2G$+2. At this point (see Figure 4b), assigning the two remaining jobs 3 and 4 in period 2 will lead to a solution that requires more than *Rlow* resource rows, because $G \geq 2$. Due to symmetry, an assignment of job 7 to row $2G$+3 will lead to the same conclusion.

***Job 7 assigned to row G*+2**: job 1 has to be assigned to rows [1, G+1] and job 2 to rows [G+3, $2G$+3]. Now, for job 4 we choose the bottom half of the schedule, rows [G+3, $2G$+3]. At this point, there are two possible ways to assign job 3 and 8 in period 2: job 8 is assigned either to row 1 or to row G+1. Considering job 8 and 5 in combination, job 8 can only be assigned to row 1 or 2. So, row 1 is the only option for job 8 and job 3 will use rows [2, G+1]. This leaves one remaining period without a job assigned to it for row 1. So, the next assignments will be: job 6 to [1,2,G+2], job 9 to $2G$+3 and job 5 to rows [2, $2G$+2]. The resulting schedule is depicted in Figure 4b. Note that an assignment of job 4 to the upper part leads to an analogous solution.

So in all solutions with R^*=*Rlow*=$2G$+3, job 7 is in the middle and the other jobs are placed such that column 3 leaves open two separate gaps of G adjacent resource units. □

Corollary 6 The optimization version of ARS-R has instances with $R^* > Rlow$.

Proof Extend instance ARS-R9(G) by adding two jobs of one period in period 3, one with demand G-1 and the other with demand G+1. This gives an instance with $R^* > Rlow$. □

By choosing an instance of ARS-R with $R^*>Rlow$ we can falsify the (Tetris-) rule of reduction of R as suggested in the previous section. Suppose that we add to the instance with $R^*>Rlow$ in each column t a new job j_t with $a(j_t)=b(j_t)=t$ and $n(j_t)$=1. This addition does not raise R^*: the assumption $R^* > Rlow$ implies that the original optimal solution provides space for job j_t in each column t. However, when reducing all the jobs j_t out of the problem, one would wrongly raise R^* by one.

3.2 NP-completeness

With proposition 5 on ARS-R9(G), we are ready to prove that the decision version of problem ARS-R is NP-complete. First, it is clear that decision problem ARS-R lies in class NP. Second, using ARS-R9(G) we will show that the NP-complete problem PARTITION can be transformed to ARS-R.

> **PARTITION** Given an indexed set of n natural numbers: $\{g_i \mid i=1,2, \ldots, n\}$, one must decide whether there exists a selection $I \subset \{1,2, \ldots, n\}$ such that $\sum_{i \in I} g_i = \frac{1}{2}\sum_{i=1}^{n} g_i$.

Given an instance of partition, define $G = \frac{1}{2}\sum_{i=1}^{n} g_i$; we may assume that G is integer. We will produce an equivalent instance of ARS by extending ARS-R9(G). For i=1,2, …, n add jobs $j=i+9$ with $n(i+9)=g_i$, such that all these jobs compete for place in period t=3, i.e., $a(i+9)=b(i+9)=3$. Replacing the partitioning question for an equivalent ARS-R question, one must decide whether only $R^* = Rlow = 2G+3$ resource rows suffice for the $n+9$ jobs. Proposition 5 tells us that this is possible only if job 7 is positioned in the middle and the other jobs are placed such that there are in column 3 two separate gaps of G adjacent units. So, the ARS-R instance will have $R^* = Rlow$ if and only if a partition of $\{g_i \mid i=1,2,\ldots,n\}$ exists. This proves proposition 7.

Proposition 7 The decision problem ARS-R is NP-complete.

In section 2.4 we saw that problem ARS-R is easy when all jobs have unary demand, $n(j)=1$, or when all jobs have unary time length, $|I(j)|=1$. The applied transformation from PARTITION to ARS-R, shows that ARS-R remains NP-complete when some of the jobs have unary demand and the other jobs unary time-length (see Figure 4).

Furthermore ARS9(G) contains jobs of only two different lengths. A problem like P | size$_j$ | C_{max} is solvable in polynomial time if all jobs require either 1 processor, or k processors (k being fixed). We can conclude for ARS-R that such cases are not solvable in polynomial time.

As it contains ARS-R as a special case, problem ARS-V is NP-complete as well.

3.3 Strongly NP-completeness of ARS-V

Problem ARS belongs to the class of strongly NP-complete problems if it is highly unlikely that an algorithm exists with running time polynomial in the numbers J, R, and/or T (more precisely, the existence of such an algorithm would be equivalent to $\mathcal{P} = \mathcal{NP}$). One can prove strongly NP-

completeness by means of a polynomial-time (or pseudopolynomial-time) reduction of some known strongly NP-complete problem to ARS. For scheduling problems the strongly NP-complete problem 3-PARTITION is often exploited.

3-PARTITION Given are numbers G and g_i with $G/4<g_i<G/2$ for $i=1,2,...,3n$ (and $\sum_i g_i = nG$). The problem asks: is there a partition $\bigcup_{k=1}^{n} I_k = \{1,2,...,3n\}$ s.t. $\sum_{i\in I_k} g_i = G$ and $|I_k| = 3$ for $k=1,2,...,n$?

One cannot transform 3-PARTITION to ARS-R by stacking copies of the solution of ARS-R9(G). E.g., to solve instances of 3-PARTITION with n=8, one cannot exploit ARS-R36(G), the ARS-R instance with 36 jobs formed with 4 copies of ARS-R9(G), to be placed in time interval [1,5] within $8G$+12 resource rows. The difficulty here is that ARS-R36(G) has alternative optimal solutions with 'holes' in column 3 of size different than G.

With this straightforward technique of stacking identical components, one on top of another in the same time-interval, we can however prove that ARS-V is strongly NP-complete.

Proposition 8 The decision problem ARS-V is strongly NP-complete, even with $T \leq 2$.

a) DATA for ARS-V3n(G) with Rlow = (G+4) n

Type of job \ t	1	2	# of copies
1	*G*+2	0	*n*
2	1	2	2*n*

b) SOLUTION: n components

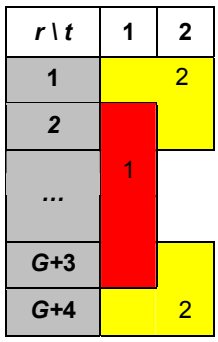

Figure 5 Instance *ARS-V*3n(G) and its partial solution

Proof Consider an instance of 3-PARTITION. We transform this instance in polynomial time to an ARS-V instance. With n copies of an ARS-V instance with three jobs, such as pictured in Figure 5a; we first generate an instance named ARS-V3n(G). Introducing in it the 3-partitioning instance, we add $3n$ jobs to ARS-V3n(G) in column t=2, that is jobs $j=3n+1$, $3n+2$, ..., $3n+3n$ with $n(j) = g_{j-3n}$ and $a(j)=b(j)=2$. We claim that the 3-PARTITION instance is a yes-instance if and only if $R^* =$ Rlow. It suffices to show that ARS-V3n(G) has a unique

solution with $R^* = Rlow$ (= $(G+4)n$) by stacking n basic-blocks, see Figure 5b, such that there are n holes of size G in t=2.

The argument is as follows. To obtain $R^* = Rlow$, all cells of column t=1 are to be covered, in other words: there cannot be a row with jobs in only period 2. Therefore, the additional demand for type-2 jobs in period 2 should take resource units previously assigned to type-1 jobs. This can only be accomplished by placing two type-2 jobs on both sides of each type-1 job, resulting in a solution as given in Figure 5b. So, the solution that stacks n of these components is the unique solution of ARS-V3n(G) with $R^* = Rlow$. Finally this solution has n holes of size G in column t=2. □

In the above proof the instance of ARS-V that results from an instance of 3-PARTITION belongs to ARS*mono*; this shows that ARS*mono* with $T \leq 2$ is strongly NP-complete as well.

3.4 Strongly NP-completeness of ARS-R

We here return to the question whether ARS-R is strongly NP-complete or solvable in pseudo-polynomial time. As mentioned in the previous section, an instance with n copies of ARS-R9(G) does not have a unique solution with $2n$ holes of size G. Therefore, we exploit the structure of ARS-R9(G) in a different way. The proof that the solution of ARS-R9(G) is unique remains valid if the horizontal jobs 7,8,9 lengthen their size to bridge a hole-width H' larger than H=1. An instance ARS-R9(G',H') with G'=2G+3 and H'=H+4 will have in its optimal solution holes, just large enough, to contain two copies of ARS-R9(G,); see Figure 6.

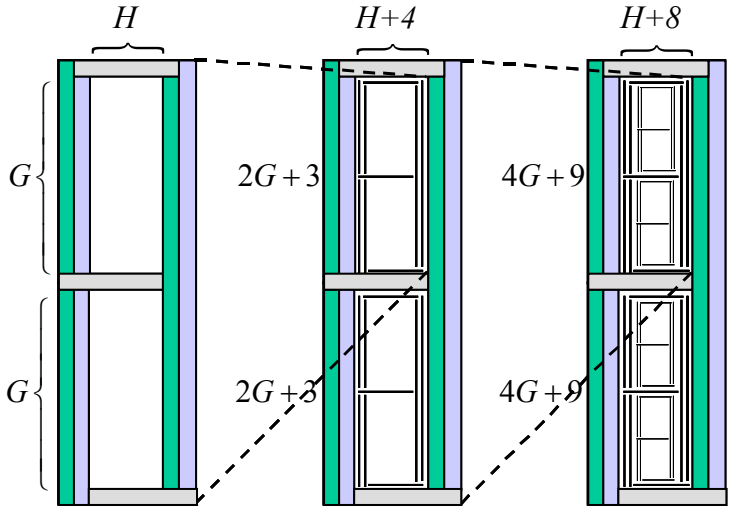

Figure 6 Schematic construction of *MOTHER(3,G,H)*

More generally, we define this way, the instance MOTHER(k,G,H) in which ARS-R9 repeats itself in k different sizes (like a Russian Matrioshka Doll): 2^{k-1} sub-instances ARS-R9(G,H), 2^{k-2} sub-instances ARS-R9($2G+3,\ H+4$), and so on, up to the single instance of ARS-R9(G',H') with $G'=2^{k-1}(G+3)-3$ and $H'=H+4(k-1)$, whereby sub-instances of level k have two holes starting at column $t=2k+1$.

Proposition 9 An optimal solution of MOTHER(k,G,H) with $G \geq 2$ and $H \geq 1$, has a series of 2^k holes of size $G \times H$ in an interval of H columns starting at $t=2k+1$.

Proof Giving a proof by induction, we first observe that the statement is true for $k = 1$. Assume that it is true for some $k \geq 1$ in combination with any $G \geq 2$ and any $H \geq 1$. Now, consider an instance MOTHER($k+1,G,H$) and some G, H. The job set in MOTHER($k+1,G,H$) consists of the job set of MOTHER($k,2G+3,H+4$) plus 2^k copies of the job set ARS-R9(G,H) (where this latter job set is shifted $2k$ periods to the right). Applying the induction hypothesis to the jobs of MOTHER($k,2G+3,H+4$) we know already that the optimal solution for this instance leaves $n=2^k$ holes of size $(2G+3)\times(H+4)$ starting at $t=2k+1$. In the first of these columns we are to place $2n$ vertical jobs with demand $G+1$ (the jobs labeled as 1 and 2 in Figure 4). Each hole must receive precisely two of such jobs: in a hole of $2G+3$ rows more than two pieces with $G+1$ rows is impossible; if some hole takes zero or one of these pieces than some other of the n holes would have to take more than two.

With two jobs of type 1 or 2 in each hole, there must be precisely one job of type 7 (Figure 4) in each hole to fill up column $t=2k+1$ completely. Similarly one can argue that each of the n holes also receives just one job of the other types (3,4,5,6,8 and 9 in Figure 4). With these 9 jobs in each hole of size $(2G+3)\times(H+4)$, proposition 5 implies that the optimal solution has a series of 2^{k+1} holes of size $G \times H$ starting at $t=2k+1+2 = 2(k+1)+1$, which completes the proof. □

We now complete the proof of strongly NP-completeness of ARS-R. Assume that there is given an instance I of 3-partition with $3n$ numbers g_i and total sum $\Sigma_{i=1,..,3n}\ g_i=nG$. Firstly, construct the instance MOTHER($k,G,1$) for the number $k=\lceil {}^2\log n \rceil$. This construction takes polynomial time $O(|I|^2)$. The resulting instance has a series of 2^k holes in column 2k+1. Secondly, we add I to MOTHER($k,G,1$) in the form of new jobs j with $a(j)=b(j)=2k+1$; for each i, a job is added with resource demand g_i in that way. Because $2n > 2^k \geq n$, we likewise add a number $d=2^k-n$ of 'dummy' jobs, each with resource demand G, to fill up d holes, if necessary, when $d>0$. Proposition 9 tells us that the final instance of ARS-R admits a schedule with $R^*=2^k(G+3)-3$, if and only if the jobs

corresponding with the numbers of I can be partitioned into triples to fit in the n holes of column $t=2k+1$.

3.5 Other special cases

The transformation from PARTITION to ARS-R of section 3.2 maps to instances of ARS-R with exactly T=5 periods, whereas proposition 4 tells us that ARS-R[T_0] is solvable in linear time for $T_0 \leq 3$. What is the complexity of ARS-R[4], the problem with planning horizon length $T = 4$?

With time horizon T=4 a method of using normal schedules gets difficult. Consider the instance ARS-R8 of ARS-R[4] with 8 jobs as given in Figure 7. Basically, there are only two optimal solutions as given (without changing the size of the holes one can interchange [2,3] with [2,2] in the right solution). When one adds to the instance a job j=9 with $a(j)=b(j)$=3 and $n(j)$=4 one cannot solve this new instance with $R^*=Rlow$. Interestingly, we were unable to create a slightly altered instance of ARS-R[4] such that each solution with $R^*=Rlow$ must have equally sized holes in one column. Therefore, we cannot easily transform PARTITION to ARS-R[4].

job j	1: 1,1	2: 1,2	3: 1,3	4: 2,2	5: 2,3	6: 2,4	7: 3,4	8: 4,4	9: 3,3
n(j)	5	2	1	3	1	1	1	6	4
n'(j)	G+2	G	1	G	1	1	1	2G+2	

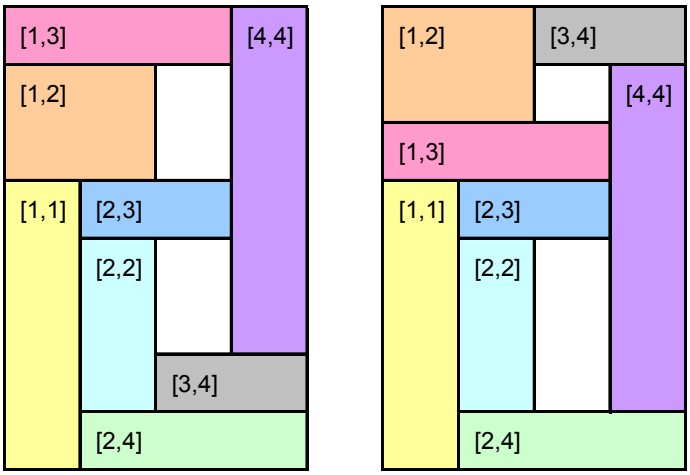

Figure 7 The two solutions of ARS-R8 can both not accommodate job 9 with $R^*=Rlow$

Nevertheless ARS-R[4] is NP-complete: with alternative resource needs for ARS-R8, given as *n'(j)* in Figure 7, we have the same optimal solution patterns as pictured, but this time both solutions have the same pair of holes of size G and G-1. We can successfully transform to this type of instance the partitioning problem 'ODD-PARTITION', which seeks for n numbers h_i with odd sum $\Sigma_i h_i = 2H$-1 a division with subsums H and H-1. Like PARTITION itself also ODD-PARTITION is NP-complete. (One

can transform an instance $\{g_i \mid i=1,2,..,n\}$ of PARTITION to the odd version by taking $h_i=2g_i$ for $i=1,2,...,n$ and $h_{n+1}=1$).

The instance ARS-R8 contains jobs with lengths 1, 2 and 3. Now consider the special case of ARS-R, where all jobs have a length of one or two periods ($\forall j : b(j) \leq a(j)+1$). This problem is solvable in linear time, with $R^*=Rlow$, by a method similar as given for ARS-R(3) in section 2.4:

1: Determine *Rlow*

2: **for** t=1 to T-1 **do**

Assign all the 2-period jobs with $a(j)=t$ and $b(j)=t+1$ (if any), in alternating fashion,

to upper rows starting from $r=1$ (if t is odd), or to lower rows starting at $r=Rlow$ (if t even).

3: Assign the 1-period jobs to the vacant rows in between the two period jobs

We have seen that ARS-R is NP-complete for instances with T fixed at a value ≥ 4. On the other hand, as we will point out below, a polynomial case arises when R is fixed. The class of instances with $Rlow \leq N$, where is N a fixed number, forms a polynomially solvable case. More generally, let subproblem ARS-R_N contain all instances with the property that in any period $t \in [1,T]$ at most N jobs are active simultaneously, i.e., $|A(t)| \leq N$ for any t. Analogously we define ARS-V_N.

Proposition 10 ARS-R_N (and also ARS-V_N) can be solved in time O(J).

Proof We present the proof only for ARS-R_N, a proof for ARS-V_N is analogous.

A dynamic programming algorithm is applied; its phases correspond to periods that are *relevant*, that is, where one or more new jobs are to be scheduled. The algorithm considers in each relevant period t a new dynamic programming state for each sequence of the jobs active in that period. Between two states in consecutive phases a link exists if and only the corresponding job sequences are compatible; this can be checked in $O(N)$ time.

Consider a link (S1, S2), where S1 is a sequencing of the jobs active in period t_1 and S2 a compatible sequencing of the jobs active at t_2. The cost of the link is the minimum number of additionally required resource units, if any, to realize the state-transition to the sequence S2 in time t_2 (if jobs of S1 that are finished at period t_2 , this cost can be zero). With t_1 and t_2 as successive relevant periods, no other jobs j (not present in S1 or S2) start in the interval (t_1,t_2); thereby one can derive the cost of each link in $O(N)$ time.

There are at most $|J|$ phases, each having at most $N!$ states, a bounded number by the assumption that N is fixed. This way dynamic programming has running time O(J), albeit with a very high coefficient as there are O($N!$) links to consider in O($N!$) states. □

Finally, we return to the problem P | fix_j, p_j=1, $|\mu_j|$=2, load=3 | C_{max}=3, the (strongly) NP-complete multiprocessing problem of section 2.2; let us label this problem for short as Q. After swapping the dimension of t and r, the instances of optimization problem ARS-R_N would ly in problem P | fix_j, load$\leq N$ | C_{max}, where N is some fixed number. In other words proposition 10 says that a full relaxation of the additional restrictions in Q (as compared to P | fix_j | C_{max}) leads to a problem solvable in polynomial time, provided that one adds the restriction: each dedicated set μ_j is an 'interval' of consecutively numbered processors.

4. Conclusion

We briefly described some applications of adjacent resource scheduling (ARS). We explored the nature of problem ARS, uncovering similarities and differences with known problems of multi-processor scheduling and two-dimensional placement. The basic problem ARS-R, with jobs as rectangles, is a special case of the P | fix | C_{max} a multiprocessor problem. Consequently the finite version of the problem, ARS-R[T_0], inherits a PTAS. The requirement of adjacency in the input data of P | fix_j | C_{max} seems somewhat unnatural in the context of computing applications, but it opens new applications of a different flavor. This may be the reason that ARS-R has not been studied so far as an individual scheduling problem.

With a linear programming model we observed for ARS-R that the minimum number of resource units R^* is very often equal to the lower bound *Rlow*. However, an associated schedule may be hard to find and we showed that R^*=*Rlow* is not necessarily the case. ARS-R[T_0], with time horizon $T_0 \leq 3$, is solvable with R^*=*Rlow* by a normal schedule. The set of normal schedules need not contain an optimal schedule for $T \geq 4$. The decision version of ARS-R is strongly NP-complete.

A second problem, ARS-V, is more general as it lets the resource demand of jobs vary in time, and resources must be assigned without unnecessary changes. In practice the latter constraint minimizes the nervousness of a system, that is, the overall change in resource occupation over time. Problem ARS-V is not a special case of multiprocessing scheduling; this problem is already strongly NP-complete for T=2.

Both ARS-V and ARS-R are polynomially solvable by dynamic programming when in each period the number of active jobs is below a fixed number N.

A subject of future research can be the existence of a PTAS for ARS-V. Further, we wonder if the problem ARS-R[T_0], with bounded time horizon T=T_0, allows a pseudopolynomial algorithm.

Acknowledgement

We are grateful to Michael Pinedo for inspiring comments made during his visit to the University of Amsterdam. We also are grateful to three anonymous referees. Due to their effort the presentation has been enhanced and they brought to our attention the relation with multiprocessor scheduling.

References

A.K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, *Scheduling independent multiprocessor tasks*, in Proceedings of the 5th Annual European Symposium on Algorithms, Graz, Austria, Lecture Notes in Computer Science 1284, Springer-Verlag, Berlin, 1–12 (1997).

J. Blazewicz, M. Drozdowski, and J. Weglarz, *Scheduling multiprocessor tasks to minimize schedule lengths,* IEEE Transactions on Computers*.* **C-35***,* 389-393 (1986).

J. Blazewicz, P. Dell'Olmo, M. Drozdowski, and M. Speranza, *Scheduling multiprocessor tasks on three dedicated processors*, Information Processing Letters **41**, 275–280 (1992) and (*Erratum*) Information Processing Letters **49**, 269-270 (1994).

J. Blazewicz, K.H.Ecker, E.Pesch, G. Schmidt, J. Weglarz, *Scheduling Computer and Manufacturing Processes*, Springer Verlag, Berlin (2001).

Chen B., C.N. Potts and G.J. Woeginger, "A review of machine scheduling: Complexity, Algorithms, and Approximability", in *Handbook of Combinatorial Optimization,* D.Z. Du and P. Pardalos (eds.), 21-169, Kluwer Academic Press, Boston (1998).

J. Chen and A. Miranda **,** *A polynomial time approximation scheme for general multiprocessor job scheduling*, SIAM Journal on computing, 1-17 (2001).

Chun H.N., *Scheduling as a multidimensional Placement Problem*, Engineering Applications of Artificial Intelligence **9**, 261-273 (1996).

Drozdowski M., *Multiprocessor Scheduling: An Overview*, European Journal of Operational Research **94**, 215-230 (1996).

J. Du and J. Y.-T. Leung, *Complexity of scheduling parallel task systems*, SIAM J. Discrete Mathematics **2**, 473–487 (1989).

Dyckhoff H., *A typology of cutting and packing problems*, European Journal of Operational Research **44**, 145-159 (1990).

Holyer I., *The NP-completeness of edge-coloring*, SIAM Journal on Computing. **10**, 718-720 (1981).

J. A. Hoogeveen, S. L. van de Velde, and B. Veltman, *Complexity of scheduling multiprocessor tasks with prespecified processor allocations*, Discrete Applied Mathematics **55**, 259–272 (1994).

Kubale M., *The complexity of scheduling independent two-processor tasks on dedicated processors,* Information Processing Letters **24**, 141-147 (1987).

Lawler E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D. Shmoys, "Sequencing and Scheduling: Algorithms and complexity", in *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, S.S. Graves, A.H.G Rinnooy Kan, and P. Zipkin, (eds.), 445-522, North Holland, New York (1993).

Lodi A., S. Martello and M. Monaci, *Two-dimensional packing problems: A survey*, European Journal of Operational Research **141**, 241-252 (2002).

Pinedo M. and X. Chao, "Operations Scheduling with applications in manufacturing and services", Irwin/McGraw-Hill, New York (1999).

Pinedo M., "Scheduling: Theory, Algorithms, and Systems", Prentice Hall, New Jersey (2001).