SCHEDULING WITH RELEASE TIMES AND DEADLINES ON A MINIMUM NUMBER OF MACHINES *

Mark Cieliebak[1], Thomas Erlebach[2], Fabian Hennecke[1], Birgitta Weber[1], and Peter Widmayer[1]

[1] *Institute of Theoretical Computer Science, ETH Zurich, 8092 Zurich, Switzerland*
cieliebak, hennecke, weberb, widmayer @inf.ethz.ch

[2] *Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland*
erlebach@tik.ee.ethz.ch

Abstract In this paper we study the SRDM problem motivated by a variety of practical applications. We are given n jobs with integer release times, deadlines, and processing times. The goal is to find a non-preemptive schedule such that all jobs meet their deadlines and the number of machines needed to process all jobs is minimum. If all jobs have equal release times and equal deadlines, SRDM is the classical bin packing problem, which is $\mathcal{NP}$-complete. The *slack* of a job is the difference between its release time and the last possible time it may be started while still meeting its deadline. We show that instances consisting of jobs with slack at most one can be solved efficiently. We close the resulting gap by showing that the problem already becomes $\mathcal{NP}$-complete if slacks up to 2 are allowed. Additionally, we consider several variants of the SRDM problem and provide exact and approximation algorithms.

1. Introduction

In this paper we study the SCHEDULING WITH RELEASE TIMES AND DEADLINES ON A MINIMUM NUMBER OF MACHINES (SRDM) problem: Given n jobs, each associated with a release time, a deadline, and a processing time, what is the minimum number of identical machines that a non-preemptive schedule needs such that all jobs meet their deadlines?

The task to process all given jobs within certain time frames and minimize the number of needed machines has recently gained new interest [5] and applies to many

* Work partially supported by the EU Thematic Network APPOL II, IST-2001-32007, with funding provided by the Swiss Federal Office for Education and Science.

practical applications. For example, consider a workshop with a variable number of repairmen. In the morning the boss gets a number of requests from customers. Each customer has a certain time window in which a repairman is allowed to visit. If there is no traveling time between customers, the SRDM problem is equal to finding the minimum number of repairmen needed for this day. If all time windows are equal, SRDM is the classical bin packing problem [8, 10]. On the other hand, if the time window of each customer is exactly the repair time, the number of needed repairmen is the same as the clique number of the corresponding interval graph.

The variant of SRDM where the goal is to decide whether all jobs can be scheduled on one machine is known as "sequencing with release times and deadlines". It is strongly $\mathcal{NP}$-complete [10]. This implies that there cannot be an approximation algorithm for SRDM with ratio $2 - \varepsilon$ for any $\varepsilon > 0$.

Related Work Machine scheduling problems have been the subject of extensive research and numerous publications (see [11] for references). Recently two variants of machine scheduling problems have gained a lot of interest: real-time scheduling [1–3], and the job interval selection problem (JISP) [6, 9, 14]. For the real-time scheduling problem the input consists of n jobs and k machines. Each of the jobs is associated with a release time, a deadline, a weight, and a processing time on each of the machines. The goal is to find a non-preemptive schedule that maximizes the sum of the weights of the jobs that meet their deadlines. The input of the JISP consists of a set of n jobs and an integer value m. Each job consists of a number of intervals on the real line. The goal is to select a subset of intervals with maximum cardinality such that at most one interval is selected for each job, and for any point x on the real line at most m intervals containing x are selected. An optimum schedule for these two problems in general just processes a subset of all jobs.

For the real-time scheduling problem constant approximation algorithms are known. In [2] Bar-Noy et al. presented an LP-based approach, whereas in [1] and [3] Bar-Noy et al., and Berman and DasGupta proposed combinatorial algorithms. If the number of machines for the JISP is one, Spieksma proved in [14] MAXSNP-hardness for this problem and proved that a greedy algorithm gives a 2-approximation. In [6] Chuzhoy et al. presented an $e/(e-1)$-approximation algorithm for JISP.

Very recently Chuzhoy and Naor [5] have studied the machine minimization problem for sets of alternative processing intervals. The input consists of n jobs and each of them is associated with a set of time intervals. A job is scheduled by choosing one of its intervals. The objective is to schedule all jobs on a minimum number of machines. Chuzhoy and Naor [5] have shown that their machine minimization problem is $\Omega(\log \log n)$-hard to approximate unless $\mathcal{NP} \subseteq DTIME(n^{O(\log \log \log n)})$.

Model and Notation Each job of the input is associated with a release time r, a deadline d, and a processing time p, where r, d, and p are integers and $d - r \geq p > 0$. The interval $[r, d)$ is the window in which an interval of size p will be placed. If the size $d - r$ of the window is equal to p, the job occupies the whole window. If the window of a job is larger than its processing time, the choice of a schedule implies for each considered job shifting an interval (the processing interval) into a position within a larger interval (the window). Therefore, we use the notation of interval graphs and *shiftable intervals* [12] $J = \langle r, d, p \rangle$. The difference $\delta = d - r - p$ is the *slack* and corresponds to the maximum amount the interval can be moved within its window. The *flexibility* of an interval in its window is described by the ratio $\tau = \frac{d-r}{p}$.

For every interval we have to select a legitimate position within its window. This position is described by a *placement* $\phi \in \{0, .., \delta\}$. The processing interval according to a placement ϕ is denoted by $J^\phi = [r + \phi, r + \phi + p)$. The range within the window that the interval has to occupy for every placement is the *core*. If the slack is smaller than the processing time, the core is the interval $[d - p, r + p)$, otherwise the core is empty.

For an n-tuple $\mathcal{S} = (J_1, .., J_n)$ of shiftable intervals, $\Phi = (\phi_1, .., \phi_n)$ defines a placement, where for $1 \leq i \leq n$ the value ϕ_i is the placement of the shiftable interval J_i. Both $\mathcal{S}$ and Φ together describe a finite collection of intervals $\mathcal{S}^\Phi = \{J_i^{\phi_i} \mid i = 1, .., n\}$ and can be interpreted as an interval graph G. For the definition of interval graphs see [4]. Since one machine can process only one job at a time, the maximum number of overlapping intervals corresponds to the minimum number of machines needed to process all jobs. This value is equal to the size of a maximum clique of the interval graph G and can be determined by a sweepline algorithm in time $O(n \log n)$.

The *domain* D of the input is the interval $[r_{\min}, d_{\max})$, where $r_{\min}$ is the earliest release time and $d_{\max}$ is the latest deadline. Let $h(\mathcal{S}^\Phi, x)$ be the number of overlapping intervals at point $x \in D$. The maximum number of overlapping intervals over all possible $x \in D$ is the *height of* $\mathcal{S}^\Phi$, which is denoted by $h(\mathcal{S}^\Phi)$. We denote the *minimum height* over all placements by $\widehat{h}(\mathcal{S})$.

The SCHEDULING WITH RELEASE TIMES AND DEADLINES ON A MINIMUM NUMBER OF MACHINES (SRDM) problem is defined as follows:

INSTANCE:	An n-tuple $\mathcal{S}$ of shiftable intervals.
SOLUTION:	A placement Φ.
MEASURE:	The height of the interval set $\mathcal{S}^\Phi$.

The decision version of the problem is SRDM(m). An instance is a yes-instance, if and only if a placement Φ exists such that the height of the corresponding set of intervals is less or equal to m.

The interval graph of all non-empty cores of the shiftable intervals in $\mathcal{S}$ is the *core graph*. Analogously, the *window graph* is the interval graph of the windows of $\mathcal{S}$. The maximum cliques of the core and window graphs obviously determine lower and upper bounds for $\widehat{h}(\mathcal{S})$.

In this paper we will present algorithms which use the *maximum slack* ($\delta_{\max}$) and *maximum flexibility* ($\tau_{\max}$) over all shiftable intervals. The *height function of a placement* Φ *for* $\mathcal{S}$ is a function $D \to \mathbb{N}_0$, where $x \in D$ is mapped to the number of intervals of $\mathcal{S}^\Phi$ which contain the point x.

Our Contribution In this paper, we give several exact and approximation algorithms for the SRDM problem and special cases of it. We start with presenting exact algorithms for the SRDM problem. In Section 2.1 we give a polynomial time algorithm for instances with $\delta_{\max} = 1$. Then we develop two dynamic programs for the decision version SRDM(m). The first one considers instances where the maximum slack is smaller than the minimum processing time. Its running time is exponential in m. The second can be used for any instance. Its running time is exponential in the maximum number of overlapping windows.

In Section 3 we describe several approximation algorithms. We explain how filling machine by machine leads to a $\Theta(\log n)$-approximation to SRDM. For restricted instances we develop algorithms with a constant approximation ratio. We show that for small windows even an arbitrary placement is a good approximation. The Greedy Best Fit algorithm is an asymptotic 9-approximation for instances with equal processing times. This algorithm can be extended for instances with a restricted ratio of processing times. For the general case we show that the number of machines determined by this algorithm can differ from the optimum value by a factor of $\Omega(\log n)$. In addition we study a few special cases. We present an asymptotic 4.62-approximation algorithm for SRDM instances where all jobs have equal release times. If the window graph is a clique we present an asymptotic 14.9-approximation.

Since the SRDM problem is easy to solve if $\delta_{\max}$ is 0 or 1, we aim to understand instances where the slack is bounded. We will prove in Section 4 that the problem is already $\mathcal{NP}$-hard for instances with every fixed $\delta_{\max} \geq 2$.

A full version of this paper can be found in [7].

2. Efficient Solutions for Special Cases

Although the SRDM problem is $\mathcal{NP}$-hard in general, some problem instances can be solved efficiently. In this section we propose a polynomial time algorithm for cases with $\delta_{\max} = 1$ and we present two dynamic programs for restricted instances of SRDM. The first approach deals with instances with small slack compared to the processing times, whereas the second has a polynomial running time if the maximum number of overlapping windows is constant.

2.1 A Polynomial Time Algorithm for SRDM with Maximum Slack 1

Next we present a polynomial time algorithm for SRDM instances with $\delta_{\max} = 1$.

THEOREM 1 *The* SRDM *problem with maximum slack at most* 1 *can be solved in polynomial time.*

Proof. We solve the decision version SRDM(m) and use binary search to determine the minimum value for m. Let the input instance $\mathcal{S} = (J_1, .., J_n)$ with domain D contain k shiftable intervals with slack 1, and $n - k$ shiftable intervals with slack 0. If the height of the cores is greater than m, SRDM(m) is a no-instance.

The placement of a shiftable interval $J = \langle r, d, p \rangle$ with slack 1 is either 0 or 1. Hence, the corresponding interval contains either the point r ($\phi = 0$) or $d - 1$ ($\phi = 1$). This observation leads to the following network flow formulation. Initially, the network contains nodes s and q, representing the source and the sink. For every shiftable interval J_i in $\mathcal{S}$ $(1 \leq i \leq n)$ with slack 1 we add a node s_i. Next we introduce a node q_x for every integer value $x \in D$ where the number of overlapping windows at point x is strictly larger than the number of overlapping cores at x. The number of q_x is at most $2k$. The source s is connected by an edge of capacity 1 to all nodes s_i representing a shiftable interval $\langle r_i, d_i, d_i - r_i - 1\rangle$. The node s_i has two outgoing edges, having capacity 1, to the nodes q_{r_i} and $q_{d_i - 1}$. There is an edge from every q_x to the sink q. Its capacity is the difference between m and the height of the cores at x.

A flow on edge (s_i, q_x) determines a unique placement of the shiftable interval J_i such that it contains x. The capacity on edge (q_x, q) guarantees that $h(\mathcal{S}^{\Phi}, x)$ is at most m for all $x \in D$. Thus, a maximum flow of size k from s to q returns a placement Φ for the input $\mathcal{S}$ such that $h(\mathcal{S}^{\Phi}) \leq m$. If the maximum flow is less than k, $\widehat{h}(\mathcal{S})$ is greater than m. The decision version $\mathrm{SRDM}(m)$ can be solved in time $O(n^2 \log n)$ using the maximum flow algorithm presented by Sleator and Tarjan [13]. ■

2.2 Dynamic Program for Rather Stiff Instances

In the following we only consider instances $\mathcal{S}$ for $\mathrm{SRDM}(m)$ where the maximum slack $\delta_{\max}$ is less than the minimum processing time $p_{\min}$. For those instances the sequence of the jobs on one machine is already determined by their release dates. Thus, if the shiftable intervals $(\langle r_1, d_1, p_1\rangle, \ldots, \langle r_n, d_n, p_n\rangle)$ are ordered by non-decreasing r_i values, then it is possible to schedule the first k jobs on m machines if and only if the first $k-1$ jobs can be scheduled in such a way that there exists at least one machine with enough remaining idle time to process the k-th job afterward.

We recursively compute table F, where $F_k(t_1, \ldots, t_m)$ indicates for $k \geq 1$ whether it is possible to schedule the first k jobs such that for all j, $1 \leq j \leq m$, machine j finishes its last job no later than t_j. We start with $F_0(t_1, \ldots, t_m) = \texttt{TRUE}$ for all integers $t_1, \ldots, t_m$ within the domain and recursively define

$$
\begin{aligned}
&F_k(t_1, \ldots, t_m) = \\
&\quad \bigvee_{j=1}^{m} ((F_{k-1}(t_1, \ldots, t_{j-1}, t_j - p_k, t_{j+1}, \ldots, t_m) \quad \wedge \quad r_k + p_k \leq t_j \leq d_k) \\
&\qquad \vee (F_{k-1}(t_1, \ldots, t_{j-1}, d_k - p_k, t_{j+1}, \ldots, t_m) \quad \wedge \quad t_j > d_k)).
\end{aligned}
$$

The value of $F_n(d_{\max}, \ldots, d_{\max})$, where $d_{\max}$ is the right endpoint of the domain, indicates if all jobs can be scheduled on m machines. If L denotes the width of the domain of the given shiftable intervals, the total effort to calculate the whole table is $O(n \cdot L^m \cdot m)$, where the last factor m results from evaluating the right hand side of the recursion above. So we have the following theorem:

THEOREM 2 *For an instance of* $\mathrm{SRDM}(m)$ *with* $\delta_{\max} < p_{\min}$ *there exists a dynamic program that computes the optimum solution with running time* $O(n \cdot L^m \cdot m)$*, where* L *denotes the width of the domain of the instance.*

2.3 Dynamic Program for Bounded Number of Overlapping Windows

THEOREM 3 *The problem* $\mathrm{SRDM}(m)$ *can be solved in time* $O(n \cdot (\delta_{\max} + 1)^H \cdot H \log H)$*, where* H *is the maximum number of overlapping windows and* n *the number of shiftable intervals.*

Proof. W.l.o.g. we assume that the window graph for a given n-tuple $\mathcal{S}$ of shiftable intervals $\langle r_1, d_1, p_1\rangle, \ldots, \langle r_n, d_n, p_n\rangle$ is connected. Let $\{t_1, \ldots, t_s\}$ be the sorted set of all distinct left window endpoints, with $s \leq n$. For $i = 1, .., s$ let $S_i = \{j \mid r_j \leq t_i < d_j\} \subseteq \{1, .., n\}$ be the set of all indices of shiftable intervals whose windows contain t_i. A local placement P for t_i is a mapping $S_i \to \mathbb{N}_0$ where $P(j) \leq d_j - r_j - p_j$ describes job j's placement.

We say that a local placement P for t_i is feasible if the resulting height of the local placement is at most m, and either $i = 1$ or there exists a feasible local placement Q for t_{i-1} such that $Q(j) = P(j)$ for all $j \in S_{i-1} \cap S_i$.

The program checks for increasing $i = 1, .., s$ the feasibility of all possible local placements for t_i. The information which is relevant for the next step is stored in a table. Assume there exists a feasible local placement P_s for t_s. If $s = 1$ then P_s is a placement of all jobs in $\mathcal{S}$. Otherwise there exists a feasible local placement P_{s-1} for t_{s-1} such that P_s and P_{s-1} place the jobs with indices from $S_{s-1} \cap S_s$ in the same way. Iteratively we know there exists a sequence $P_1, .., P_s$ of feasible placements for $t_1, .., t_s$ which represents a placement for all jobs. On the other hand, if we are given a placement Φ for the input $\mathcal{S}$ such that $h(\mathcal{S}^\Phi) \leq m$, the restriction of Φ to S_s gives a feasible local placement for t_s. Details of the running time analysis are omitted due to space restrictions. ■

3. Approximation Algorithms for SRDM

In this section, we develop and analyze approximate solutions. We start with an intuitive approach of iteratively filling machines. We show that an arbitrary placement is a good approximation if $\frac{\delta_{\max}}{p_{\min}}$ is small. We develop the Greedy Best Fit algorithm, which is a good approximation for instances where the processing times do not differ very much. A lower bound for this algorithm is presented as well. Finally, we study approximation algorithms for instances where all windows have one common point.

3.1 A $\Theta(\log n)$-Approximation Algorithm

A greedy 2-approximation algorithm for the job interval selection problem (JISP) is presented in [14]. To solve the SRDM problem we use this algorithm and successively load machines with jobs until no job is left over.

THEOREM 4 *Iteratively filling the machines using a constant approximation algorithm for JISP leads to a $\Theta(\log n)$-approximation algorithm for* SRDM.

3.2 Instances with Small Windows

If the ratio of the maximum slack $\delta_{\max}$ and the minimum processing time $p_{\min}$ is small, an arbitrary placement of all shiftable intervals is already a good approximation for SRDM.

THEOREM 5 *An arbitrary placement is a $(k+1)$-approximation for* SRDM*, where* $k = \lceil \frac{2\delta_{\max}}{p_{\min}} \rceil$.

Proof. The proof is by contradiction. Assume that, for an n-tuple $\mathcal{S}$ of shiftable intervals with $\delta_{\max} \leq \frac{k}{2} \cdot p_{\min}$, there exists a placement Φ and a point x such that $h(\mathcal{S}^\Phi, x) \geq (k+1) \cdot \widehat{h}(\mathcal{S}) + 1$.

Let $\mathcal{T} \subseteq \mathcal{S}$ be the subset of shiftable intervals whose windows contain x. Using an optimum solution, this subset can be partitioned into at most $\widehat{h}(\mathcal{S})$ sets $\mathcal{T}_1, .., \mathcal{T}_t$ such that the height of $\mathcal{T}_i$ is 1. By Φ_i for $i = 1, .., t$ we denote the restriction of Φ to the set $\mathcal{T}_i$. Using an averaging argument we know that for the placement Φ there exists one $v \in \{1, .., t\}$ with $h(\mathcal{T}_v^{\Phi_v}, x) \geq k + 2$. The number s of elements in $\mathcal{T}_v$ must be at least $k + 2$. Consider a placement Ψ of the shiftable intervals in $\mathcal{T}_v$ with $h(\mathcal{T}_v^{\Psi}) = 1$. W.l.o.g. we assume the elements in $\mathcal{T}_v$ are sorted such that $r_i + \psi_i < r_{i+1} + \psi_{i+1}$ for $i = 1, .., s - 1$. We consider the points $L = r_1 + p_1 + \psi_1 \geq d_1 - \delta_1$ and $R = r_s + \psi_s \leq r_s + \delta_s$. By definition of $\mathcal{T}_v$ at least k intervals can be placed between L and R, hence $R - L \geq k \cdot p_{\min}$. Since all windows in $\mathcal{T}_v$ contain x we know $d_1 > x \geq r_s$. We obtain $k \cdot p_{\min} \leq R - L \leq r_s + \delta_s - (d_1 - \delta_1) < \delta_s + \delta_1 \leq 2 \cdot \delta_{\max}$, which contradicts our hypothesis. ■

3.3 The Greedy Best Fit Algorithm

If the ratio between the longest and shortest processing time is bounded we propose the Greedy Best Fit (GBF) algorithm. This algorithm processes the shiftable intervals $\mathcal{S}$ in order of increasing window size. For a job J the algorithm calculates for every placement ϕ the maximum height inside the interval J^{ϕ}. From the subset of placements which lead to the lowest height, it chooses the leftmost.

$\mathcal{I} = \emptyset$
for $J = \langle r, d, p \rangle \in \mathcal{S}$ in order of increasing window size **do**
  for $\phi = 0, .., r - d - p$ **do** $h_{\max}[\phi] = \max_{x \in J^{\phi}} h(\mathcal{I}, x)$ **end for**
  $h_{\min} = \min\{h_{\max}[\phi] \mid \phi = 0, .., r - d - p\}$
  $\psi_J = \min\{\phi \mid \phi = 0, .., r - d - p$ and $h_{\max}[\phi] = h_{\min}\}$
  add J^{ψ_J} to $\mathcal{I}$
end for

Algorithm 1: GBF-Algorithm

For the analysis of the GBF algorithm, we define the *work* of a set of intervals I in $[a, b]$ as the value $\int_a^b h(I, x)\, dx$. We start with the case where all processing times are equal.

THEOREM 6 *For a* SRDM *instance $\mathcal{S}$ with equal processing times the GBF algorithm returns a placement Φ such that $h(\mathcal{S}^{\Phi}) \leq 9 \cdot \widehat{h}(\mathcal{S}) + 1$.*

Proof. Let $\mathcal{I}$ be the intervals placed by the GBF algorithm. Denote $h = h(\mathcal{I})$. Consider the first shiftable interval $J = \langle r, d, p \rangle$ in $\mathcal{S}$ whose placement increases the height to h. Denote by $I \subseteq \mathcal{I}$ the set of all intervals which have been placed so far, not including J, i.e. $h(I) = h - 1$ and $h(\{J^{\phi}\} \cup I) = h$ for all placements ϕ. The size of J's window is $L = d - r$. Let $I' \subseteq I$ denote the subset of intervals whose intersection with $[r, d)$ is non-empty and denote by W the work of I' in $[r, d]$. Since the size of the windows of all intervals in I' is not greater than L, even an optimal solution has to place them between $r - L$ and $d + L$. It follows $\widehat{h}(\mathcal{S}) \geq \frac{W}{3L}$.

To obtain a lower bound on W, we construct a set of intervals I'' with the following properties: the height of I'' is $h - 1$, any placement of J increases the height to

h, and the work of I'' is minimal. Since the placement of J increases the height, there is no range of length p between r and d with height at most $h-2$. Hence, there must be peaks of height $h-1$ at least every $p-1$ points within the interval $[r, d]$. Thus I'' consists of $N = \left\lceil \frac{L-(p-1)}{2p-1} \right\rceil$ peaks of height $h-1$ and width p and we get a lower bound $W \geq N \cdot (h-1) \cdot p$. From the definition of N we have $L \leq N \cdot (2p-1) + p - 1 \leq (2 \cdot N + 1) \cdot p$. Since N is a positive integer, with the bounds on L and W we have $\widehat{h}(\mathcal{S}) \geq \frac{W}{3L} \geq \frac{N(h-1)}{3 \cdot (2N+1)} \geq \frac{h-1}{9}$ ■

The idea of the proof above can be extended to instances with different processing times by constructing the peaks of intervals of size $p_{\max}$ and calculating their work as if they were of size $p_{\min}$. This results in

THEOREM 7 *GBF has asymptotic approximation ratio* $9 \cdot \frac{p_{\max}}{p_{\min}}$ *for* SRDM.

If we partition a given instance of SRDM into subinstances such that the ratios between the maximal and the minimal processing times are bounded by $e \approx 2.718$, we obtain

THEOREM 8 *There exists an asymptotic* $9e \cdot \left\lceil \ln\left(\frac{p_{\max}}{p_{\min}}\right) \right\rceil$*-approximation algorithm to* SRDM.

The GBF algorithm can be implemented using 3 nested for loops. The way we presented it in Algorithm 1 its running time depends on the size of the domain. With some changes the algorithm can be implemented in time $O(n^3)$.

Lower Bound for Greedy Best Fit. Unfortunately for general instances the GBF algorithm does not have a constant approximation ratio.

THEOREM 9 *The height of the GBF algorithm can differ from the optimum by a factor of* $\Omega(\log n)$.

3.4 Constant Approximation for Complete Window Graphs

If the release times of all shiftable intervals are equal and all deadlines are equal as well, we have the classical bin packing problem ([8, 10]). For this problem constant approximation algorithms and asymptotic polynomial time approximation schemes are known. Hence it would be interesting to generalize these results to SRDM instances with a complete window graph.

Equal Release Times. In the following we investigate the case where all release times are 0 but the deadlines differ.

THEOREM 10 *If all release times of shiftable intervals are equal the Divide Best Fit algorithm returns a placement* Φ *with* $h(\mathcal{S}^{\Phi}) \leq \frac{1}{2}(7+\sqrt{5}) \cdot \widehat{h}(\mathcal{S}) + 1$.

Proof. The Divide Best Fit algorithm splits $\mathcal{S}$ into two sets according to their flexibility values. The set $\mathcal{S}_f$ denotes relatively flexible shiftable intervals where the τ values are at least τ_0. The remaining shiftable intervals $\mathcal{S}_s$ are relatively stiff.

For $\tau_0 = \frac{1}{2} \cdot (1 + \sqrt{5})$ partition the given instance $\mathcal{S}$ into
$\mathcal{S}_f = \{\langle 0, d, p\rangle \in \mathcal{S} \mid \frac{d}{p} \geq \tau_0\}$ and $\mathcal{S}_s = \{\langle 0, d, p\rangle \in \mathcal{S} \mid \frac{d}{p} < \tau_0\}$.
Use the GBF algorithm for $\mathcal{S}_f$.
for $J \in \mathcal{S}_s$ **do** place J at the rightmost position **end for**

Algorithm 2: Divide Best Fit Algorithm

For the flexible set $\mathcal{S}_f$ the GBF algorithm traverses the shiftable intervals by increasing right endpoints. It places the intervals as far to the left as possible such that the height is minimized. This algorithm leads to a collection of intervals $\mathcal{I}_f$ with height ALG_f. It is not difficult to see that $\widehat{h}(\mathcal{S}_f)$ can be bounded by $\widehat{h}(\mathcal{S}_f) \geq \frac{\tau_0 - 1}{\tau_0} \cdot (\mathrm{ALG}_f - 1)$. In the stiff set $\mathcal{S}_s$ all intervals are placed at their rightmost position. The resulting height is at most twice the optimum height.Using the lower bound on $\widehat{h}(S_f)$ we obtain the stated approximation. ■

Obviously, the Divide Best Fit algorithm can be adapted to solve problem instances where the release times differ and all deadlines are equal.

Window Graph Is a Clique. Next we want to generalize the problem discussed in the previous section and consider instances $\mathcal{S} = (\langle r_1, d_1, p_1\rangle, \ldots, \langle r_n, d_n, p_n\rangle)$ of SRDM where all windows have a common point x. Thus, it holds $r_i \leq x < d_i$ for $1 \leq i \leq n$. We partition $\mathcal{S}$ into three disjoint subsets $\mathcal{S}_L$, $\mathcal{S}_R$ and $\mathcal{S}_C$. The set $\mathcal{S}_L$ contains all members J_i of $\mathcal{S}$ whose cores do not contain x and for which the part of the window left of x is larger than the part to the right of x ($x - r_i > d_i - x$). Similarly, the set $\mathcal{S}_R$ contains all members J_i of $\mathcal{S}$ whose cores do not contain x and for which $x - r_i \leq d_i - x$. The remaining shiftable intervals are in $\mathcal{S}_C$ and have cores overlapping in x. We transform $\mathcal{S}_R$ into $\mathcal{S}_{R|}$ by setting all release times to x, and $\mathcal{S}_L$ into $\mathcal{S}_{L|}$ by setting all deadlines to x. Now we use the Divide Best Fit algorithm to place $\mathcal{S}_{L|}$ and $\mathcal{S}_{R|}$ independently. Finally, we place the intervals in $\mathcal{S}_C$ arbitrarily. To analyze the approximation ratio of the described algorithm, we first show that the height of the optimum solution for $\mathcal{S}_{R|}$ is at most three times the optimal height for $\mathcal{S}_R$.

LEMMA 11 *The minimum height of $\mathcal{S}_{R|}$ can be bounded by $\widehat{h}(\mathcal{S}_{R|}) \leq 3\widehat{h}(\mathcal{S}_R)$.*

Proof. We change the optimum placement for $\mathcal{S}_R$ such that the resulting placement is feasible for the restricted instance $\mathcal{S}_{R|}$ and its height only increases by a factor of three. We change the placements for all shiftable intervals placed to the left of x. If an interval is placed completely to the left of x we replace it with its mirror image where the mirror is at x. This is feasible since the part of the window to the right of x is larger than its remaining part. This operation, carried out for all intervals to which it applies, increases the height of the placement by a factor of at most 2. If the interval is placed such that it contains x, it is shifted to the right of x. This shifting can increase the height by another $\widehat{h}(\mathcal{S}_R)$. We have a new placement where all intervals are placed to the right of x and its height is at most $3 \cdot \widehat{h}(\mathcal{S}_R)$. ■

The analogous result holds for $\mathcal{S}_{L|}$, too. The height of the optimal placement for $\mathcal{S}$ is at least the minimum height for every single set $\mathcal{S}_L$, $\mathcal{S}_R$, and S_C. Using

Theorem 10, the placement computed by the algorithm for $\mathcal{S}_{R|}$ has height at most $\frac{1}{2}(7+\sqrt{5})\cdot 3\widehat{h}(\mathcal{S}_R)+1$, and similarly for $\mathcal{S}_{L|}$. Since the domains of $\mathcal{S}_{L|}$ and $\mathcal{S}_{R|}$ are non-overlapping, the height of the overall solution computed by the algorithm is at most $\frac{1}{2}(7+\sqrt{5})\cdot\max\{3\widehat{h}(\mathcal{S}_L), 3\widehat{h}(\mathcal{S}_R)\}+1+|\mathcal{S}_C|\leq\frac{1}{2}(23+3\sqrt{5})\cdot\widehat{h}(\mathcal{S})+1$. This gives the following theorem.

THEOREM 12 *For* SRDM *instances $\mathcal{S}$ where the window graph is a clique, there exists an approximation algorithm such that the resulting height is at most $\frac{1}{2}(23+3\sqrt{5})\widehat{h}(\mathcal{S})+1$.*

4. $\mathcal{NP}$-Hardness of SRDM with Maximum Slack 2

As shown in Section 2.1, the SRDM problem is easy to solve if the maximum slack is at most 1. Furthermore a very simple approximation algorithm can be found if the ratio between maximum slack and minimum processing time is small. Hence, it seems as if small slack makes the problem easy. Surprisingly, already if $\delta_{\max}=2$ the SRDM problem is $\mathcal{NP}$-hard.

THEOREM 13 *The* SRDM *problem with $\delta_{\max}=2$ is $\mathcal{NP}$-hard.*

Proof. This proof is by reduction from 3-SAT [10]. The input is a set U of variables and a Boolean formula $\mathcal{F}$ in conjunctive normal form. Every clause C contains 3 literals, where a literal is a variable or a negated variable in U. 3-SAT asks for an assignment to U such that $\mathcal{F}$ is satisfied. For a Boolean formula $\mathcal{F}$ having k clauses, we construct a set of shiftable intervals $\mathcal{S}$ such that $\widehat{h}(\mathcal{S})=3k$ if and only if $\mathcal{F}$ is satisfiable. W.l.o.g. we assume that every variable in U occurs in $\mathcal{F}$ negated and not negated.

The construction of $\mathcal{S}$ contains generators for variables, clause gadgets and gadgets for copying values of literals. We are going to explain our construction with help of Figure 1.

Generator: For every variable we construct a generator $\mathcal{G}$ with starting point g. In our example they are positioned left, indicated by light-grey boxes. A generator is built out of four shiftable intervals. Two of them have a slack of zero (are *fixed*). Both shiftable intervals with slack greater zero overlap one of the fixed intervals by 1 unit. These two represent both literals of the variable. Only one of its literals can be `TRUE`. An interval in its leftmost position is interpreted as a literal set to `TRUE`. Otherwise the literal is set to `FALSE`.

$$\mathcal{G}=(\langle 0,g+1,g+1\rangle,\langle 0,g,g\rangle,\underbrace{\langle g,r_1,r_1-g-1\rangle,\langle g,r_2,r_2-g-1\rangle}_{x,\neg x})$$

The endpoints r_1, r_2 of the windows are defined by a starting point of a copy gadget, or by a clause. Since $\widehat{h}(\mathcal{G})=2$, at least one of the intervals has to be shifted to the right and represents a literal which is `FALSE`. Observe that both shiftable intervals starting at g can be shifted to the right, and thus represent false. Then the corresponding variable will not contribute to the result.

Copy Gadget for Values of Literals: In $\mathcal{F}$ a literal can occur more than once. Thus, the construction of $\mathcal{S}$ has to ensure that all shiftable intervals representing the same

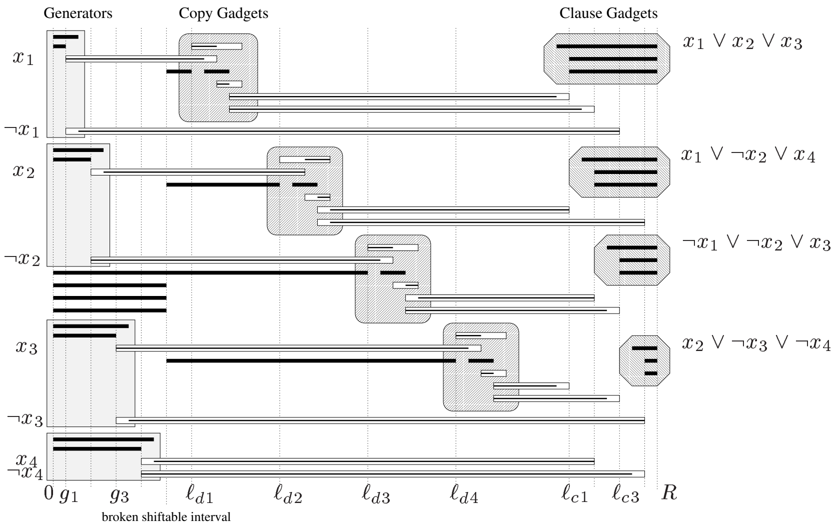

Figure 1. $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$

literal have the same value. To copy values of literals a *copy gadget* with the following form is used:

$$\mathcal{D} = (\underbrace{\langle \ell, \ell_d + 2, \ell_d - \ell + 1 \rangle}_{\text{original literal}}, \underbrace{\langle 0, \ell_d, \ell_d \rangle}_{\text{place holder}}, \langle \ell_d, \ell_d + 4, 2 \rangle, \langle \ell_d + 1, \ell_d + 3, 2 \rangle,$$
$$\langle \ell_d + 2, \ell_d + 4, 1 \rangle, \underbrace{\langle \ell_d + 3, r_1, r_1 - \ell_d - 4 \rangle, \langle \ell_d + 3, r_2, r_2 - \ell_d - 4 \rangle}_{\text{two copies}})$$

In the example these gadgets are depicted by boxes with round corners. Within the copy gadget exists only one shiftable interval with slack 2. The value ℓ_d represents the starting point of the copy gadget.

To not exceed $\widehat{h}(\mathcal{D}) = 2$ the copy gadget has to work in the following way: If the original literal is placed left, both copies can also be placed left without exceeding height 2. On the other hand, if the original interval is shifted to the right, both copies have to be shifted to the right in order to obtain the minimum height for this gadget.

Observe that for every copy gadget we introduce one shiftable interval with slack zero starting at zero. To simplify Figure 1 we split some of these intervals into two parts. It is always possible to shift an interval representing a literal to its right endpoint, even if it could also be placed left. This would set the corresponding literal to `FALSE`. The important point is: if once a literal is set to `FALSE`, all copies of this literal will represent `FALSE` as well.

The Clause Gadget: For every clause a *clause gadget* is constructed. In Figure 1 these gadgets are octagons and placed at the right end. Every gadget is built out of three shiftable intervals representing literals and three shiftable intervals with zero slack.

$$\mathcal{C} = (\underbrace{\langle \ell_1, \ell_c, \ell_c - \ell_1 - 1 \rangle, \langle \ell_2, \ell_c, \ell_c - \ell_2 - 1 \rangle, \langle \ell_3, \ell_c, \ell_c - \ell_3 - 1 \rangle}_{\text{literal intervals}},$$
$$\langle \ell_c - 1, R, R - \ell_c + 1 \rangle, \langle \ell_c, R, R - \ell_c \rangle, \langle \ell_c, R, R - \ell_c \rangle) \text{ for some } R > \ell_c$$

The starting points of the literals are defined either by a generator or by a copy gadget. To not exceed $\widehat{h}(\mathcal{C}) = 3$, at most 2 of the literals can be shifted to the right (FALSE). Hence, at least one literal must not be shifted – and represents a TRUE literal. A clause gadget $\mathcal{C}$ has its left starting point at an appropriate position ℓ_c defined by the placement. As indicated in Figure 1, the value R has to be the same value for all clauses in $\mathcal{F}$.

Placement of Components: All gadgets have to be placed independently of each other. As shown in the example, the starting points of the generators differ and the copy gadgets have their starting points one after the other without influencing each other. At the right end of the domain the clause gadgets are placed similar to the generators on the left. The formula $\mathcal{F}$ contains n different variables, k clauses, and hence $3k$ literals. If there exists a placement Φ for $\mathcal{S}$ such that $\mathcal{S}^{\Phi}$ has height $3k$, then the 3-SAT formula $\mathcal{F}$ is satisfiable. Since we placed all gadgets independently, it is essential that all generators and copy gadgets have height two, and every clause gadget has height three. If no copy gadget has height three, all literals set to FALSE at their generators are represented by a shifted interval at the corresponding clause gadget. Observe that for every placement the height at point $R - 1$ and 0 is exactly $3k$. Because at most two intervals are allowed to be shifted at every clause gadget – to not exceed the height – at least one interval of every clause gadget must not be shifted. Thus, if the Boolean formula $\mathcal{F}$ is satisfiable, a placement Φ for $\mathcal{S}$ can be found such that $h(\mathcal{S}^{\Phi})$ is $3k$. Otherwise $\widehat{h}(\mathcal{S})$ is at least $3k + 1$. ■

In the proof of Theorem 13 the maximum flexibility $\tau_{\max}$ is 2. We can adapt the gadgets and the placements of the components such that we obtain the following result:

THEOREM 14 *The* SRDM *problem is $\mathcal{NP}$-hard for arbitrary* $\tau_{\max} > 1$.

5. Conclusion

We studied the SRDM problem, a scheduling problem motivated by a variety of practical applications. We presented positive and negative results, but there are still open questions. Is there an asymptotic PTAS or an approximation algorithm with a constant approximation ratio for arbitrary problem instances? Even if we cannot hope for a $(2 - \varepsilon)$ - approximation algorithm, an asymptotic PTAS could still exist.

Acknowledgment We would like to thank Riko Jacob for many helpful comments and suggestions.

References

[1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J.S. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[2] A. Bar-Noy, S. Guha, J.S. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, 31(2):331–352, 2001.

[3] P. Berman and B. DasGupta. Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization*, 4(3):307–323, 2000.

[4] A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: a Survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.

[5] J. Chuzhoy and S. Naor. New hardness results for congestion minimization and machine scheduling. accepted for STOC'04, 2004.

[6] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. In *IEEE Symposium on Foundations of Computer Science*, pages 348–356, 2001.

[7] M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling jobs on a minimum number of machines. Technical Report 419, Institute of Theoretical Computer Science, ETH Zürich, 2003.

[8] E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS, 1996.

[9] T. Erlebach and F.C.R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *Journal of Algorithms*, 46(1):27–53, 2003.

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, New York, 1979.

[11] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science*, volume 4, pages 445–522. North-Holland, 1993.

[12] F. Malucelli and S. Nicoloso. Shiftable interval graphs. In *Proc. 6th International Conference on Graph Theory*, 2000.

[13] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[14] F.C.R. Spieksma. Approximating an interval scheduling problem. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 1444, pages 169–180. Springer-Verlag LNCS, 1998.