Competitive Algorithms for Mulitstage Online Scheduling

Michael Hopf[a,*], Clemens Thielen[a], Oliver Wendt[b]

[a] *University of Kaiserslautern, Department of Mathematics Paul-Ehrlich-Str. 14, D-67663 Kaiserslautern, Germany*

[b] *University of Kaiserslautern, Department of Economics Erwin-Schrödinger-Str. 42, D-67663 Kaiserslautern, Germany*

Abstract

We study an online flow shop scheduling problem where each job consists of several tasks that have to be completed in t different stages and the goal is to maximize the total weight of accepted jobs. The set of tasks of a job contains one task for each stage and each stage has a dedicated set of identical parallel machines corresponding to it that can only process tasks of this stage. In order to gain the weight (profit) associated with a job j, each of its tasks has to be executed between a task-specific release date and deadline subject to the constraint that all tasks of job j from stages $1, \ldots, i-1$ have to be completed before the task of the ith stage can be started. In the online version, jobs arrive over time and all information about the tasks of a job becomes available at the release date of its first task. This model can be used to describe production processes in supply chains when customer orders arrive online.

We show that even the basic version of the offline problem with a single machine in each stage, unit weights, unit processing times, and fixed execution times for all tasks (i.e., deadline minus release date equals processing time) is APX-hard. Moreover, we show that the approximation ratio of any polynomial-time approximation algorithm for this basic version of the problem must depend on the number t of stages.

For the online version of the basic problem, we provide a $(2t-1)$-competitive deterministic online algorithm and a matching lower bound. Moreover, we provide several (sometimes tight) upper and lower bounds on the competitive ratio of online algorithms for several generalizations of the basic problem involving different weights, arbitrary release dates and deadlines, different processing times of tasks, and several identical machines per stage.

Keywords: scheduling, online optimization, competitive analysis

*Corresponding author. Fax: +49 (631) 205-4737. Phone: +49 (631) 205-4824

Email addresses: hopf@mathematik.uni-kl.de (Michael Hopf), thielen@mathematik.uni-kl.de (Clemens Thielen), wendt@wiwi.uni-kl.de (Oliver Wendt)

1. Introduction

Scheduling is concerned with the allocation of jobs to scarce resources (machines). In revenue management, each job has a certain weight (or revenue) and the goal is to output a feasible subset of the jobs with maximum total weight. For each job j, the scheduler has to decide whether to accept the job (occupying a machine) or reject it (losing potential revenue). Additionally, the scheduler must decide to which machine each accepted job should be assigned and when it should be executed within the time interval between its release date r_j and deadline d_j.

The special case where the processing requirement p_j of a job j is equal to $d_j - r_j$ is known as as *interval scheduling*. Here, the scheduler does not have to decide when to execute a job since each accepted job has to start directly at its release date and will end at its deadline.

The more general case $p_j \leq d_j - r_j$ is considered as *job scheduling*, where the difference $d_j - r_j$ is called *interval length*.

In this paper, we analyze a scheduling problem where each job j consists of several tasks and each of them has to be scheduled in a certain *stage*. There is a dedicated set of parallel machines available for each stage $i \in \{1, \ldots, t\}$ and these machines can only process tasks of stage i (where each machine can process one task at a time). Each job j has one task T_j^i in each stage i and each task T_j^i has a specific release date r_j^i, processing time p_j^i and deadline d_j^i, where we assume that $d_j^{i-1} \leq r_j^i$ for $i = 2, \ldots, t$. Each job j has a nonnegative weight w_j that is obtained if job j is accepted, in which case all tasks of job j have to be completed by their deadlines on the machines of the corresponding stages. The objective is to maximize the total weight of accepted jobs.

In the online version of the problem, jobs arrive over time and all tasks of a job become known at the release date of the first task of the job. Here, the scheduler is allowed to abort previously accepted jobs in order to accept jobs arriving later (which might have larger weight).

The problem is motivated by production processes in supply chains. Our model handles several stages that can be thought of as steps of a production process. Profit is only obtained after a customer order (job) has been processed in all stages and the final product is delivered to the customer.

Another motivation is the map-reduce paradigm [1], which is a programming model for processing and generating large data sets. The idea is to portion the input into map tasks that can be run on map machines in the first stage outputting key-value pairs. In the second stage, these pairs serve as input for the reduce machines. A more detailed description of the map-reduce paradigm from a scheduling perspective can be found in [2], where the authors consider a variant of the two-stage flexible flow shop problem motivated by the map-reduce paradigm.

1.1. Previous Work

Some of our results focus on interval scheduling, i.e., the case where the processing time of each job/task is equal to its deadline minus its release date.

Single-stage interval scheduling problems are well-studied in literature. The single-stage offline version of our problem is known to be efficiently solvable in polynomial time, even in the case of arbitrary weights [3, 4]. For unit weights, even the online problem with a single stage can be solved optimally by a simple greedy algorithm [5, 6]. When different weights are allowed, however, this problem does not admit any competitive online algorithms if no further restrictions are imposed [7, 8].

A model similar to ours was considered in [9]. Here, the authors provide (offline) approximation algorithms for a single-stage model in which each job (or *t-interval*) consists of a union of at most t half-open intervals. However, there is only one machine that processes all the intervals whereas, in our model, we have a separate set of machines in each stage, which implies that tasks of different stages cannot interfere with each other. In [10], the authors consider the online selection of t-intervals, which do not necessarily arrive in order of their left endpoints, and provide upper and lower bounds on the competitive ratio of randomized online algorithms.

Other researchers consider similar problems. Bafna et al. [11] and Berman et al. [12] analyze the problem of scheduling nonoverlapping local alignments, which corresponds to our basic problem in the offline case. Their work is motivated by applications in computational molecular biology. In [11], the authors analyze the *IR problem*, which consists of choosing a maximum independent subset of axis-parallel rectangles, where two rectangles are independent if both of their projections on the axes do not intersect. They show NP-completeness of the problem even for unit weights and provide a tight analysis of a natural local improvement heuristic for general weights. In [12], the authors provide a 3-approximation for the two-dimensional weighted version that runs in $\mathcal{O}(n \log n)$ time. In [13], the authors provide inapproximability results for the independent set problem in d-box graphs, i.e., intersection graphs of axis-parallel rectangles in $\mathbb{R}^d$. Here, intersections between rectangles are defined by the intersection of sets in $\mathbb{R}^d$ rather than by considering projections to the axes.

The offline case of our problem is a special case of finding the maximum weight independent set in a d-claw free graph. More specifically, scheduling jobs in t stages can be thought of finding a maximum weight independent set in the corresponding intersection graph, which is $(2t + 1)$-claw free. Approximation algorithms for the maximum weight independent set problem in d-claw free graphs can be found in [14]. A $d/2$-approximation algorithm is given in [15]. This corresponds to a $(t + \frac{1}{2})$-approximation algorithm for our scheduling problem. In [16], the author presents an approximation algorithm for the unweighted maximum independent set problem in d-claw free graphs based on local improvement search and achieves an approximation ratio of $\frac{d-1}{2} + \epsilon$. This yields a $(t + \epsilon)$-approximation algorithm for the basic version of our scheduling problem.

To the best of our knowledge, the online case of our scheduling setting with the objective of maximizing the total weight of accepted jobs has not been studied so far.

1.2. Our Contribution

We present online algorithms for several cases of the problem, sometimes obtaining tight upper and lower bounds on the competitive ratio achievable by any online algorithm. Our competitiveness results are summarized in Table 1.

The basic problem considered in Section 3 is a restricted version with unit weights, unit processing times, unit interval lengths, and a single machine in each stage. We obtain upper and lower bounds on the competitiveness of online algorithms for this setting as well as for the generalizations to arbitrary weights (Section 4.1) and unit weights but arbitrary interval lengths (Section 4.2). Afterwards, we consider the combination of different weights and arbitrary interval lengths (Section 4.3), the generalization of the basic problem to arbitrary processing times (Section 4.4) as well as a general model combining different weights, arbitrary interval lengths, and arbitrary processing times (Section 4.5). In the section on parallel machines (Section 4.6), we analyze the problem with several identical parallel machines in each stage and show how several results from the single-machine case can be extended.

For the offline problem, we show that even the offline version of the basic problem considered in Section 3 is APX-hard. Moreover, we show that (unless $\mathsf{P} = \mathsf{NP}$) there does not exist a constant factor approximation algorithm for the basic (offline) problem when the number t of stages is considered as part of the input. Hence, even for the offline problem, the approximation ratio achieved by any polynomial-time algorithm must depend on t. The algorithms we present achieve a competitiveness linear in t even for the online problem and almost all of them run in polynomial time.

Note that, even though we allow an online algorithm to abort previously accepted jobs, the online algorithms we present only make use of this in the case of different weights of jobs. For this setting, it is easy to see that a deterministic online algorithm that does not abort jobs cannot be competitive, even on instances consisting of only two jobs and a single stage.

Problem	**Upper Bound**	**Lower Bound**
Basic	$2t-1$	$2t-1$
Arbitrary Weights	$8t-4$	$2t-1$
Interval Lengths ≤ 2	$2t-1$	$2t-1$
Arb. Interval Lengths	$2t$	$2t$
Arb. Weights + Arb. Interval Lengths	$8t+2$	$2t$
Arb. Processing Times	$\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)-1$	$\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)-1^*$
General	$4\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)+3$	any of the previous
Parallel Machines	$2t$	2

Table 1: Competitiveness results for different problems with t stages, different weights and $\Delta(i) := \max_{j,k} \frac{p_j^i}{p_k^i}$. The bound marked with an asterisk only holds for algorithms that never abort previously accepted jobs.

2. Problem Definition

We consider the problem of scheduling n jobs on machines in $t \geq 2$ stages. In each stage i, there is a set M_i of m_i identical parallel machines available. Each job j has one tasks T_j^i corresponding to it in each stage $i \in \{1, \ldots, t\}$. Tasks of stage i can only be processed on machines in M_i, i.e., the machines corresponding to stage i. Each task T_j^i of a job j has a *release date* r_j^i, a *deadline* d_j^i and a *processing time* $p_j^i > 0$, where we assume that $d_j^i \geq r_j^i + p_j^i$. We refer to $r_j := r_j^1$ as the release date of job j. The intervals $I_j^i := [r_j^i, d_j^i)$ are the intervals corresponding to the tasks T_j^i of a job j and we call $|I_j^i| = d_j^i - r_j^i$ the *interval length* of task T_j^i. Moreover, we assume that $d_j^{i-1} \leq r_j^i$ for each job j and $i = 2, \ldots, n$. This ensures that the i*th* task of job j can only be processed after the previous task has been completed. This is actual not a restriction since our algorithms also work without this assumption, but the analysis is more technical in this case. However, it is possible that $r_{j_1}^i < r_{j_2}^{i-1}$ for $j_1 \neq j_2$, i.e., it is possible that tasks of different stages are processed simultaneously if they belong to different jobs.

If a job j is accepted, each task T_j^i of job j has to be scheduled on a machine from M_i between its release date r_j^i and deadline d_j^i and we gain a *weight* (or *profit*) of $w_j \geq 0$. If a job is rejected, it is lost forever and cannot be accepted at any later point in time.

A feasible schedule consists of a set S of accepted jobs together with a starting time s_j^i with $r_j^i \leq s_j^i \leq d_j^i - p_j^i$ and a machine in M_i for each task T_j^i of each accepted job $j \in S$ such that tasks assigned to the same machine do not intersect, i.e., such that $[s_{j_1}^i, s_{j_1}^i + p_{j_1}^i) \cap [s_{j_2}^i, s_{j_2}^i + p_{j_2}^i) = \emptyset$ whenever two tasks $T_{j_1}^i$ and $T_{j_2}^i$ of the same stage i are assigned to the same machine in M_i.

The objective is to maximize the total weight (profit) of accepted jobs. In the special case of unit weights, this reduces to maximizing the number of accepted jobs.

To illustrate problem instances and schedules, we represent tasks by rectangles labeled by the number of the job they correspond to (cf. Figure 4). The thickness of the frame of the rectangle indicates the stage to which the corresponding task belongs (e.g., in Figure 4, tasks with a thin frame belong to the first stage and tasks with a thick frame to the second stage).

The main focus of our work is on the online version of the problem where jobs arrive over time, i.e., a job j including all its tasks is revealed at its release date $r_j = r_j^1$ and the scheduler has to decide immediately whether to accept the job and when to schedule all tasks of j. Here, the scheduler is allowed to abort previously accepted jobs in order to accept jobs arriving later (which might have a larger weight). If a job j is aborted, all currently executed tasks of job j are aborted, no further tasks of the job need to be executed, and the weight w_j is lost, even if some tasks of the job have already been completed. Migration of an already started task to another machine is not allowed.

Another natural possibility for the scheduler is to move tasks of already accepted jobs in later stages (that have not yet been started) in order to be able

to accept newly arriving jobs. All our competitiveness results hold true whether this is allowed or not with the only exception of the lower bound of $2t$ for the case of arbitrary interval length. However, in this case, we can still inherit the lower bound of $2t - 1$ from the basic problem.

To measure the quality of online algorithms, we use *competitive analysis* [17], where one compares, for each input sequence σ, the profit $\mathrm{ALG}(\sigma)$ obtained by an online algorithm ALG to the optimal profit $\mathrm{OPT}(\sigma)$ achievable on that sequence. A (deterministic) online algorithm ALG for a maximization problem is called *c-competitive* for a constant $c \geq 1$ if $\mathrm{ALG}(\sigma) \geq \frac{1}{c} \cdot \mathrm{OPT}(\sigma)$ for every input sequence σ. The *competitive ratio* of an online algorithm is defined as the infimum over all c such that the algorithm is c-competitive. In the following, we will usually refer to an input sequence of the problem in the online version as well as in the offline version as an *instance*.

3. The Basic Problem

In this section, we consider a restricted version of the problem, which we call the *basic problem*. Here, each job j has unit weight $w_j = 1$ and each task T_j^i has unit processing time $p_j^i = 1$ and unit interval length $d_j^i - r_j^i = 1$. There are t stages, but only one machine per stage.

We start by showing APX-hardness of the basic problem. Afterwards, we consider the online version and provide a $(2t-1)$-competitive online algorithm and a matching lower bound on the competitive ratio achievable by any online algorithm.

3.1. Complexity of the Offline Problem

Theorem 1. *The basic problem is APX-hard, even if there are only two stages.*

Proof. We provide a reduction from the maximum independent set problem in graphs of maximum vertex degree three (*degree three graphs*), which is well-known to be APX-complete [18].

Let $G = (V, E)$ be an arbitrary undirected graph with maximum vertex degree three. Without loss of generality, we assume that there are no isolated vertices in G. We construct an instance of the basic problem with two stages such that an independent set of size k in G corresponds to a schedule of profit k and vice versa.

For the construction of the instance, we use the following definitions (cf. Figure 1):

Definition 1. A *chain* is a set of tasks such that each task starts 0.5 time units after the previous one. Hence, two consecutive tasks can never be scheduled together.

Definition 2. A *2-block* consists of two tasks starting at exactly the same time.

Definition 3. A *3a-block* consists of three tasks starting at exactly the same time.

Definition 4. A *3b-block* is a chain of three tasks, i.e., the second task intersects with the first and the third task, but the first and the third task do not intersect.

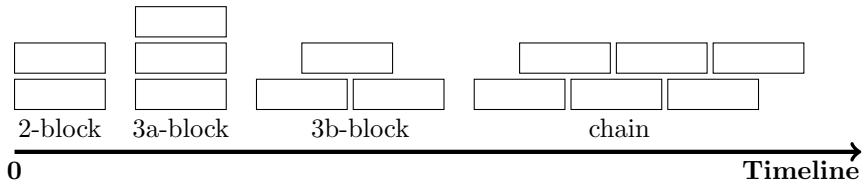

Figure 1: Illustration of blocks and a chain of jobs.

To construct our instance, we first partition G into inclusionwise maximum elementary paths (that are not cycles) by iteratively starting with an arbitrary vertex $v \in V$ and then always choosing an edge from one of the endpoints of the path to some vertex that is not yet contained in the path until no such edge exists anymore. Afterwards, we remove all vertices of the path and all edges incident to them from G and iterate the procedure.

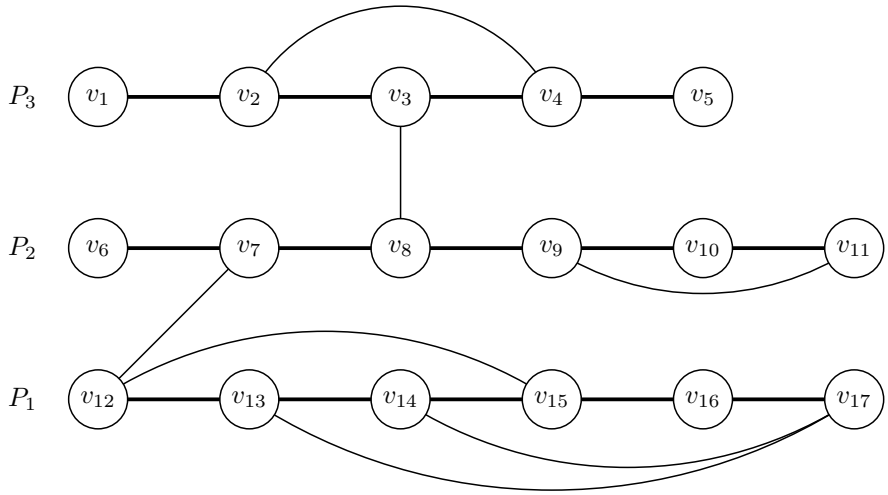

Figure 2: The partition of the graph G into elementary paths (here P_1, P_2 and P_3).

After having partitioned G into several elementary, disjoint paths $P_1, \ldots, P_k$, we create one job for each vertex $v \in V$. In the first stage, the tasks corresponding to each path P_i form a chain (cf. Figure 2) and all these chains are placed such that no task from one chain intersects with a task from another chain.

For the second stage, we consider the graph $G' = (V, E')$ obtained from G by removing all edges that are contained in the paths $P_1, \ldots, P_k$. Observe that all vertices have degree at most two in G'. We first let each vertex that has degree

zero in G' correspond to a task that does not intersect any other tasks in the second stage. Afterwards, we remove all these vertices from G' and let the tasks corresponding to each pair of adjacent vertices of degree one form a 2-block in the second stage (cf. v_3 and v_8 or v_9 and v_{11} in Figure 2). All vertices that have degree two in G' must be end vertices of paths P_i and, by the maximality of the paths P_i, no two such vertices can be adjacent in G' (except if they belong to the same path). If an end vertex of some path P_i is connected to two adjacent vertices, we let the tasks corresponding to these jobs form a 3a-block in the second stage (cf. v_{13}, v_{14}, and v_{17} in Figure 2). If an end vertex is connected to two (inner) vertices that are not adjacent (in G), these jobs are scheduled as a 3b-block, where the end vertex corresponds to the job intersecting both other jobs (cf. v_7, v_{12}, and v_{15} in Figure 2). If two end vertices v and w are adjacent in G', they have to belong to the same path. Each of them can have at most one more adjacent vertex (say v' for v and w' for w) that we have to consider. Then, we just schedule these jobs as a chain in the order v', v, w, w'. All blocks in the second stage are placed such that no task from one block intersects with a task from another block and no block intersects with any of the tasks of the second stage corresponding to the degree zero vertices of G'.

From this construction, it is immediately clear that an independent set of size k corresponds to a schedule with k jobs and vice versa. □

Corollary 2. *Unless $P = NP$, the basic problem is not approximable within a factor of* 1.0005*. In particular, there does not exist a PTAS for the problem.*

Proof. This is true for the maximum independent set problem in degree three graphs. □

Lemma 3. *Unless $P = NP$, the basic problem with t stages is not approximable within any constant factor if t is part of the input.*

Proof. Consider the following straightforward reduction from the maximum independent set problem (in general graphs): Given an instance of maximum independent set, we create one job for each vertex and one stage for each edge $e = (u, v)$. In this stage, no tasks intersect except for the tasks corresponding to u and v, which are placed as a 2-block. Then, an independent set of size k corresponds to a schedule with k jobs and vice versa and the claim follows since maximum independent set does not admit a constant factor approximation algorithm unless $\mathsf{P} = \mathsf{NP}$. □

3.2. Online Competitiveness Results

We now consider the online version of the basic problem. We provide a simple greedy algorithm and show that it is 3-competitive for two stages. Then, we show that this result and its proof generalize to yield $(2t-1)$-competitiveness of the algorithm for the case of t stages, which we prove to be best possible.

Algorithm 1 GREEDY for the basic problem

Accept a newly arriving job if it does not intersect with any previously accepted job.

The jobs arrive over time, i.e., by increasing release date in the first stage. Our algorithm GREEDY (Algorithm 1) processes them one by one and accepts each job if it does not intersect with any previously accepted job. If several jobs are released simultaneously, GREEDY processes them in an arbitrary order.

Theorem 4. *Algorithm 1 is* 3*-competitive for the basic problem with two stages.*

PROOF. For a fixed instance, let GO denote the set of jobs scheduled by GREEDY and OPT the set of jobs scheduled in a fixed optimal schedule. We define a mapping $\phi : \mathrm{OPT} \to \mathrm{GO}$ as follows (see Figure 3 for an illustration):

Case 1: For $j \in \mathrm{OPT} \cap \mathrm{GO}$, we set $\phi(j) := j$.

Case 2: If $j \in \mathrm{OPT} \setminus \mathrm{GO}$, job j was rejected at its arrival because there is at least one other job in GO that intersects with j.

Case 2.1: In the schedule of GREEDY, the machine of the first stage is busy with job k at the release date of job j. Then, we set $\phi(j) := k$. Note that we have $r_k^1 \leq r_j^1$.

Case 2.2: In the schedule of GREEDY, the machine of the first stage is idle, but the second task of j does not fit in the schedule anymore because there are one or two previously accepted jobs blocking the machine of the second stage. Among these jobs, let k be the one with the earliest release date in the second stage. Then, we set $\phi(j) := k$.

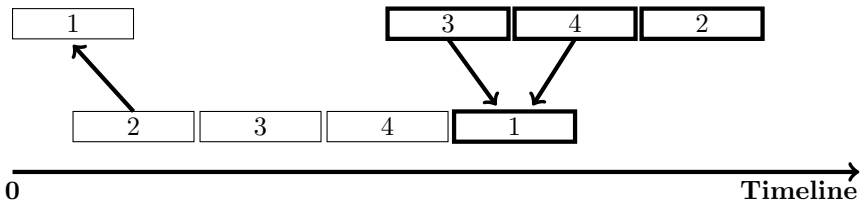

Figure 3: Example for the mapping ϕ for the 2-stage problem. Tasks of the first stage have a thin frame; tasks of the second stage have a thick frame. GREEDY just accepts the first job, whereas the optimal schedule is $\{2, 3, 4\}$. Jobs 2, 3, and 4 are all mapped to job 1 as illustrated by the arrows.

In order to show that GREEDY is 3-competitive, it suffices to show that any job in GO has at most three preimages under ϕ. First observe that each job $j \in \mathrm{OPT} \cap \mathrm{GO}$ only has itself as a preimage since, in Case 2, we only map jobs from OPT to jobs in GO that intersect with these. For the jobs in $\mathrm{GO} \setminus \mathrm{OPT}$,

observe that, due to the unit processing time of the tasks, there can be at most two tasks in OPT intersecting with one task in GO, which directly yields 4-competitiveness since we obtain at most two preimages per stage for any job in GO. However, by definition of ϕ, a job j from OPT is never mapped to a job k from GO with $r_j^1 < r_k^1$. Thus, we only obtain one preimage from the first stage, which shows that any job in GO has at most three preimages under ϕ in total. □

Observe that, if we map each job in Case 2.2 in the proof of Theorem 4 in the earliest stage in which the task of the job does not fit into the schedule anymore, we obtain at most two preimages for each job in GO in each stage $2, \dots, t$. Since the first stage only contributes one preimage as before, this directly yields the following result for t stages:

Corollary 5. *Algorithm 1 is $2t-1$-competitive for the basic problem with t stages.* □

The naive implementation of GREEDY, where we check all previously accepted jobs (at most $n-1$ many) for intersections for each incoming job, runs in $\mathcal{O}(n^2)$ time for a fixed number of stages. However, in the first stage, where jobs arrive in order of increasing release dates, it is enough to store the latest deadline of a job to check for intersections in constant time. Moreover, in any other stage, we can check for intersections more efficiently by storing the tasks corresponding to accepted jobs in the stage ordered by their release dates. We can then insert a new task and check whether it intersects with any previously accepted task of the stage in $\mathcal{O}(\log n)$ time by using binary search. In total, this yields a running time of $\mathcal{O}(n \log n)$ for a fixed number of stages.

We now show that the competitiveness of Algorithm 1 is in fact best possible:

Proposition 6. *No deterministic online algorithm for the basic problem with t stages can achieve a competitive ratio smaller than $2t-1$.*

PROOF. Consider an instance with t stages and $2t$ jobs with the following intervals corresponding to the respective tasks in the first stage (where, for simplicity, we let the unit processing time correspond to two time units): $I_1^1 = [0, 2)$, $I_2^1 = [1, 3)$ and $I_j^1 = [2j-3, 2j-1)$ for each $j = 3, \dots, 2t$. For the second stage, we let $I_1^2 = [4t-1, 4t+1)$, $I_2^2 = [4t+2, 4t+4)$, $I_3^2 = [4t-2, 4t)$, $I_4^2 = [4t, 4t+2)$, and $I_j^2 = [4t+2j-6, 4t+2j-4)$ for further jobs $j \in \{5, \dots, 2t\}$ (cf. Figure 4 for $t = 2$ stages).

For simplicity, we do not describe the release dates and deadlines of further stages explicitly. In stage $k \geq 3$, the task corresponding to job 1 is placed such that it intersects with the tasks corresponding to jobs $2k-1$ and $2k$, whereas all other tasks are placed without any intersections.

In order to be competitive, the algorithm has to accept the first job presented (otherwise, it is not competitive for the instance in which no further jobs arrive). But if an algorithm accepts the first job, no other jobs can be accepted anymore. Thus, ALG $= 1$, whereas OPT $= 2t-1$ by accepting every job but the first one.

If ALG aborts the first job in order to accept some later job l in the sequence, the adversary constructs the same instance as before based on job l, i.e., regarding job l as the new first job and presenting further jobs accordingly (shifting the release dates to the right). After some time, any competitive online algorithm has to process the first job to completion and then the adversary presents the original sequence of jobs. □

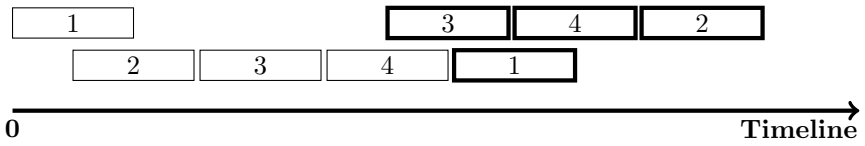

Figure 4: Example for the 2-stage problem. Tasks of the first stage have a thin frame; tasks of the second stage have a thick frame. The tasks of the first stage are given by the following intervals: $I_1^1 = [0, 2)$, $I_2^1 = [1, 3)$, $I_3^1 = [3, 5)$, and $I_4^1 = [5, 7)$. The tasks of the second stage are given by the intervals $I_1^2 = [7, 9)$, $I_2^2 = [10, 12)$, $I_3^2 = [6, 8)$, and $I_4^2 = [8, 10)$.

Theorem 4 and Proposition 6 yield the following result:

Corollary 7. *Algorithm 1 has competitive ratio* $2t - 1$ *for the basic problem with* t *stages.*

An approximation for the offline version of the basic problem can be obtained from the $(\frac{d-1}{2} + \epsilon)$-approximation algorithm for the unweighted maximum independent set problem in d-claw free graphs presented in [16], which yields a $(t + \epsilon)$-approximation algorithm for the basic problem. However, the running time of this algorithm is $\mathcal{O}(n^{\log \frac{1}{\epsilon}})$, which becomes impractical for small values of ϵ.

Similarly, the $\frac{d}{2}$-approximation algorithm for the maximum weight independent set problem in d-claw free graphs presented in [15] yields a $t + \frac{1}{2}$ approximation algorithm for the basic problem.

4. Generalizations

In this section, we study the generalizations of the basic problem to arbitrary weights, arbitrary interval lengths, and arbitrary processing times. We focus on the online problem and show how the ideas from the previous section can be generalized to obtain competitiveness results for the more general settings.

4.1. Arbitrary Weights

We first consider the generalization of the basic problem to arbitrary weights $w_j \geq 0$, where the goal is to maximize the total weight of accepted jobs. Recall that it is allowed to abort previously accepted jobs in order to accept jobs arriving later (which might have a larger weight). It can easily be seen that,

without this possibility, no deterministic online algorithm can be competitive, even on instances consisting of only two jobs and a single stage.

We analyze the following generalization of Algorithm 1:

Algorithm 2 GREEDY for the basic problem with arbitrary weights

Accept a newly arriving job j if and only if its weight w_j is strictly larger than twice the total weight of the set Int(j) of previously accepted jobs that intersect with j (aborting all jobs in Int(j)).

Note that Algorithm 2 always accepts a job with positive weight if it does not intersect with any previously accepted jobs.

Theorem 8. *Algorithm 2 is* $(8t-4)$*-competitive for the basic problem with* t *stages and arbitrary weights.*

PROOF. We analyze the class of algorithms that accept a job j if and only if its weight w_j is strictly larger than $\tau \geq 1$ times the total weight of the jobs in Int(j). Algorithm 2 is contained in this class, where we set $\tau = 2$. Indeed, we prove that this is the choice of τ that yields the best competitiveness.

For a fixed instance, let GO denote the set of jobs scheduled by GREEDY and OPT the set of jobs scheduled in a fixed optimal schedule. Let GO(z) be the set of jobs already accepted and not (yet) aborted by GREEDY at time z (possibly including a job with release date z).

This time, we define a distribution of the total weight of all jobs in OPT to the jobs in GO such that each job $j \in$ GO gets at most $(2t-1)(\tau + \tau/\tau-1)$ times its own weight assigned to it. Therefore, we first want to map the weight of a job in OPT to jobs in GO(z) for some z and then iteratively map it to jobs in GO.

Let $j \in$ OPT and $Q \subseteq$ GO(r_j) be the set of jobs in GO(r_j) that intersect with j (possibly containing j itself). By definition of GO(r_j) and our greedy rule, we can distribute the weight of j such that each job in Q gets assigned at most τ times its own weight (note that $j \in$ GO(r_j) if GREEDY accepts j at time r_j, in which case the weight of the job is mapped to j itself). However, the jobs in GO(r_j) do not necessarily have to be in GO since they could be aborted after time r_j by GREEDY.

If a job k in GO(r_j) is not in GO, there is a job l_1 that GREEDY accepted at some time while aborting k. By our greedy rule, we have $\tau w_k < w_{l_1}$. We map the weight that was originally mapped to k to the job l_1. Of course, we cannot be sure that $l_1 \in$ GO, but we can continue this procedure inductively until we finally end up with a job in GO.

We now show that each job $j \in$ GO gets assigned at most $(2t-1)(\tau + \tau/\tau-1)$ times its own weight, which proves the claim. Therefore, we distinguish two cases how a job $j \in$ GO can get weight assigned to it: from jobs that are released at or before time r_j, and from jobs that are released strictly after time r_j.

Case 1: jobs $k \in \text{OPT}$ with $r_k \leq r_j$
To analyze the first case, we define P_1 to be the set of jobs that were aborted by GREEDY in order to schedule j (which might be empty). For $i \geq 2$, we then let P_i be the set of jobs that were aborted in order to accept the jobs in P_{i-1} (cf. Figure 5). Obviously, there is no job in P_i that does not intersect with some job in P_{i-1}. Since GREEDY aborted the jobs in P_i in order to accept the jobs in P_{i-1}, it then follows that the total weights $w(P_i) := \sum_{j' \in P_i} w_{j'}$ of the jobs in the sets P_i satisfy

$$w_j > \tau w(P_1) > \tau^2 w(P_2) > \ldots > \tau^{i-1} w(P_{i-1}) > \tau^i w(P_i) > \ldots \quad . \qquad (1)$$

We know that the jobs in P_1 have total weight at most w_j/τ, and the weight of the jobs in P_i have weight at most $w(P_{i-1})/\tau$, or at most w_j/τ^i.

Now, let $k \in \text{OPT}$ be a job with $r_k \leq r_j$ for which some of its weight is finally assigned to j. Then, by definition of the weight assignment, this weight must first be mapped to jobs in $\text{GO}(r_k)$, each of which is contained in some set P_i. In particular, this part of the weight of k is directly mapped only to jobs in the sets P_i that intersect with k and have release date smaller or equal to r_k.

Thus, due to the unit length of all tasks, there can be at most $(2t-1)$ many jobs of OPT that directly map weight to one job q in P_i, and by definition of the weight assignment, from each such job $l \in \text{OPT}$, job q is assigned weight of at most τw_q. Hence, the weight directly mapped to the jobs in P_i in total is at most

$$(2t-1)\tau w(P_i) \overset{(1)}{<} (2t-1)\tau^{-i+1} w_j.$$

Thus, the total weight that is mapped from the jobs of OPT to j over the jobs in P_i is at most $(2t-1)\tau^{-i+1} w_j$. In total, if we consider all sets P_i, $i \geq 1$, we obtain that a job $j \in \text{GO}$ can be assigned total weight of at most

$$\sum_{i=1}^{\infty} (2t-1)\tau^{-i+1} w_j \leq (2t-1) w_j \sum_{i=0}^{\infty} \tau^{-i} = (2t-1) w_j \frac{\tau}{\tau - 1}$$

from jobs of OPT that are released up to time r_j.

Case 2: jobs $k \in \text{OPT}$ with $r_k > r_j$
Now, we regard the case of jobs of OPT that were released strictly after time r_j. Due to the unit lengths of all tasks, there can be at most $(2t-1)$ jobs $k \in \text{OPT}$ with $r_j < r_k$ that intersect with j. Hence, since each job $k \in \text{OPT}$ with $r_j < r_k$ that (directly) assigns weight to j, assigns weight at most τw_j to j and was rejected by GREEDY directly at its arrival (so it can never have been assigned any weight of other jobs), job j is assigned at most $(2t-1)\tau w_j$ weight in total from jobs with later release date than r_j.

Summing up:
Summing up both cases, at most

$$(2t-1)\frac{\tau}{\tau - 1} w_j + (2t-1)\tau w_j = (2t-1) w_j \left(\frac{\tau}{\tau - 1} + \tau \right)$$

weight is mapped to a job $j \in \mathrm{GO}$ in total as claimed.

As the function $f(\tau) = \frac{\tau}{\tau-1} + \tau$ attains its minimum at $\tau = 2$, the choice $\tau = 2$ as in Algorithm 2 yields the best competitiveness of $(2t-1) \cdot 4 = (8t-4)$.

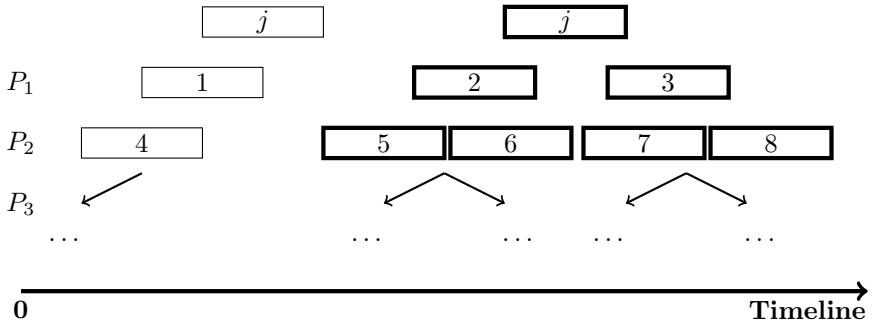

Figure 5: Visualization of the sets P_i for two stages. The tasks of the first stage have a thin frame; the tasks of the second stage have a thick frame.

□

We remark that Theorem 8 generalizes a result of Woeginger [8], who showed that the single-stage version of Algorithm 2 is 4-competitive.

4.2. Arbitrary Interval Lengths

In this section, we again consider the basic problem with unit weights, but we allow $d_j^i - r_j^i > p_j^i$, i.e., the interval lengths are arbitrary (and possibly different for each task of a job). In this setting, an algorithm has to decide for each job j whether to accept the job and, if it is accepted, when to execute each corresponding task T_j^i within the time interval $I_j^i = [r_j^i, d_j^i)$. Although the additional decision of task placement seems to make the problem much more complex, we now show that similar competitiveness results as for the basic problem with unit interval lengths can still be obtained.

We analyze the following greedy algorithm:

Algorithm 3 GREEDY for the basic problem with arbitrary interval lengths

1: Accept a newly arriving job if it is possible to schedule all of its tasks in their corresponding intervals given the previous placement of the tasks of the already accepted jobs.
2: If a job is accepted, schedule each of is tasks as early as possible.

GREEDY processes the jobs one by one in order of their arrival (i.e., by increasing release date in the first stage) and always accepts the next job if all of its tasks can be feasibly scheduled between their release date and deadline given the previous placement of the tasks of the already accepted jobs. If a job j

is accepted, each of its tasks T_j^i is started as early as possible after its release date r_j^i.

Theorem 9. *Algorithm 3 is $2t$-competitive for the basic problem with arbitrary interval lengths and t stages. In the case where all intervals have length at most two, Algorithm 3 is $(2t-1)$-competitive.*

PROOF. The proof is very similar to the one of Theorem 4. Again letting GO and OPT denote the set of jobs accepted by GREEDY and in a fixed optimal schedule, respectively, we now define a mapping $\phi : \mathrm{OPT} \to \mathrm{GO}$ as follows:

Case 1: For $j \in \mathrm{OPT} \cap \mathrm{GO}$, we set $\phi(j) := j$.

Case 2: If $j \in \mathrm{OPT} \backslash \mathrm{GO}$, job j was rejected by GREEDY at its arrival because some of its tasks could not be scheduled anymore within their corresponding intervals. Let i be the earliest stage in which task T_j^i cannot be scheduled by GREEDY and let s_j^i denote the starting time of task T_j^i in the optimal schedule. We then set $\phi(j) := k$, where k is such that T_k^i is the task started earliest by GREEDY among the tasks that are (partly) executed by GREEDY within the time interval $[s_j^i, s_j^i + 1)$ (at least one such task must exist since otherwise GREEDY could have scheduled task T_j^i in the time interval $[s_j^i, s_j^i + 1)$).

Since all tasks have unit processing times and, in Case 2, we only map jobs from the optimal schedule to jobs in the schedule computed by GREEDY that intersect with these, it follows as in the proof of Theorem 4 that no job $j \in \mathrm{GO}$ can have more than two preimages per stage from Case 2 and at most one preimage from the first stage. Hence, we obtain at most $2t - 1$ preimages for a job in GO from Case 2 and at most one preimage (the job itself) from Case 1, which shows that no job has more than $2t$ preimages in total. Note that, other than in the proof of Theorem 4, Cases 1 and 2 are not mutually exclusive anymore, i.e., a job $j \in \mathrm{OPT} \cap \mathrm{GO}$ can have itself as preimage plus up to $2t-1$ additional preimages from Case 2 (see Figure 6).

To see that GREEDY is $2t-1$-competitive if all intervals have length at most two, note that, in this case, a job $j \in \mathrm{OPT} \cap \mathrm{GO}$ can have at most one preimage in each stage from Case 2: If $j \in \mathrm{GO}$ has two preimages k, l in some stage i from Case 2, both tasks T_k^i and T_l^i are scheduled in the optimal schedule such that they intersect the (unit length) time interval in which GREEDY executes task T_j^i. In particular, in the optimal schedule, one of the two tasks starts strictly before GREEDY starts task T_j^i and the other one ends strictly later than GREEDY finishes task T_j^i (compare Figure 6). Since the interval length of task T_j^i is at most two, this implies that T_j^i cannot be scheduled in the optimal schedule, so $j \notin \mathrm{OPT}$. □

Again, we can show that the competitiveness of our greedy algorithm is in fact best possible:

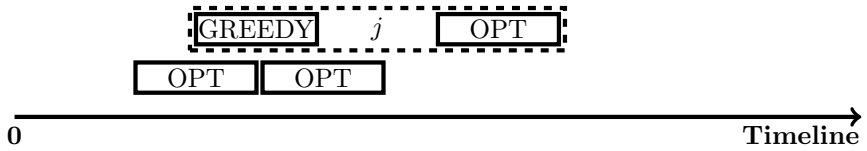

Figure 6: The figure shows the tasks of three jobs in a stage $i \geq 2$. GREEDY accepts job j and places its task at the beginning of the corresponding interval, which is denoted by the dashed lines. In the optimal schedule, the task of job j is scheduled at the end of the interval. Now, two other jobs arrive whose tasks in stage i have interval length one and intersect with job j where GREEDY placed it. Hence, these two jobs cannot be accepted by GREEDY anymore and might become preimages of job j under ϕ even though $j \in \text{OPT} \cap \text{GO}$.

Proposition 10. *No deterministic online algorithm for the basic problem with arbitrary interval lengths and t stages can achieve a competitive ratio smaller than $2t$.*

PROOF. Let ALG be an arbitrary online algorithm and consider the following sequence of jobs: At the beginning, a job 1 with an interval length of 5 in all stages arrives. If ALG does not accept job 1, it is not competitive since we can end the sequence of jobs after job 1. Otherwise, we continue the sequence exactly as in the proof of Proposition 6 (cf. Figure 4 for $t = 2$ stages) with the position of the task of job 1 in each stage being the position at which ALG scheduled the task. Hence, ALG will only be able to accept job 1, while an optimal offline algorithm can accept all $2t - 1$ addtional jobs presented. Moreover, due to the interval length of 5 of job 1 in all stages, an optimal offline algorithm can additionally schedule the task corresponding to job 1 in each stage $k \geq 2$ either before or after the tasks corresponding to jobs $2k-1$ and $2k$ that intersect with the task of job 1 (as scheduled by ALG) in stage k. Hence, by scheduling the first task of job 1 at a different point in time than ALG (which is always possible due to the interval length of 5 if we leave some space between the tasks of job 2 and 3 in the first stage), an optimal offline algorithm can also accept job 1, so it can schedule $2t$ jobs in total. □

4.3. Arbitrary Interval Lengths and Arbitrary Weights

If we allow the jobs to have arbitrary interval lengths and arbitrary weights, our greedy approach is to schedule a job as early as possible, possibly aborting other jobs if they have weight less than a certain fraction of the weight of the incoming job (see Algorithm 4).

Algorithm 4 GREEDY for the problem with arbitrary interval lengths and weights

1: Let $\tau := 1 + \sqrt{1 + 1/2t}$.
2: If (given the previous placement of the tasks of the already accepted jobs) it is possible to schedule all tasks of a newly arriving job j without aborting any previously accepted jobs, do so (scheduling tasks as early as possible).
3: Otherwise, compute a minimum weight set $\mathrm{Int}(j)$ of already accepted jobs whose abortion makes it possible to schedule all tasks of job j. Accept j aborting the jobs in $\mathrm{Int}(j)$ if the weight of job j is strictly larger than τ times the total weight of the jobs in $\mathrm{Int}(j)$, scheduling all tasks of j as early as possible.

There are three possibilities what GREEDY can do at the release date r_j of a newly arriving job j:

1. Schedule j somewhere without aborting any other job.
2. Schedule j aborting other jobs that have total weight less than w_j/τ. If there are several possibilities for a set of previously accepted jobs to abort, we choose an arbitrary set.
3. Reject j because no matter where we would schedule the tasks of j, we had to abort jobs of total weight at least w_j/τ.

Note that it can be shown that it is NP-hard to find a feasible abortion placement of the incoming job as in step 2 of Algorithm 4. However, if we do so, GREEDY is $(8t + 2)$-competitive.

Theorem 11. *Algorithm 4 is $(8t + 2)$-competitive for the basic problem with arbitrary interval lengths, arbitrary weights, and t stages.*

PROOF. The proof works analogously to the proof of Theorem 8 since we start mapping the jobs to each other after all of them are placed, i.e., we do not have to consider different possibilities for the placement of tasks within their intervals at this time. As before, we consider an algorithm that accepts a job j if jobs of total weight less than w_j/τ have to be aborted, where $\tau \geq 1$ is arbitrary.

Recall that we say that two jobs j and l intersect if, in some stage $k \in \{1, \ldots, t\}$, they are placed at $[s_j^k, s_j^k + 1)$ and $[s_l^k, s_l^k + 1)$ and the intersection of these intervals is nonempty.

Again, let $\mathrm{GO}(z)$ be the set of jobs that are already accepted and not (yet) aborted by GREEDY at time z. Let $j \in \mathrm{OPT}$. We define $Q \subseteq \mathrm{GO}(r_j)$ to be the set of jobs in $\mathrm{GO}(r_j)$ that intersect with j (with respect to j's placement in the optimal schedule) including the job j itself if $j \in \mathrm{GO}(r_j)$.

We first map the weight of a job $j \in \mathrm{OPT}$ to the jobs in Q. Note that, if j was accepted by GREEDY at time r_j, the weight can be mapped to j itself (as $j \in Q$ in this case). Otherwise, we can conclude by our greedy rule that the

jobs in Q have total weight at least w_j/τ and, thus, we can distribute w_j to the jobs in Q such that each job gets assigned at most τ times its own weight.

If a job $k \in Q$ is not in GO, there is a job l_1 that GREEDY accepted at some time while aborting k with $\tau w_k < w_{l_1}$. We map the weight mapped to k to the job l_1 and continue this procedure inductively until we end up with a job in GO.

Now let $j \in$ GO. We bound the weight assigned to j by $(2t\tau + 1)\frac{\tau}{\tau-1}w_j$, which is at most $(8t + 2)w_j$ for the choice of τ in the algorithm.

Similar to the proof of Theorem 8, we take a look at the jobs in P_i that were aborted in order to schedule the jobs in P_{i-1} (with $P_0 := \{j\}$). Again, we get $w(P_i) < \tau^{-i}w_j$.

If a job $k \in$ OPT finally assigns weight to j, it first assigns some of it to some job $l \in P_i$, but at most τw_l. Due to the unit lengths of jobs, there can be at most $2t$ jobs of OPT that intersect with l. They can each assign τw_l weight to l (this is different to the case of unit interval length since a job k of OPT can now map to a job q of GO(r_k) if the starting time of k in the first stage in the schedule of OPT is earlier than the starting time of q in the schedule of GREEDY). Hence, each job $l \in P_i$ contributes a total weight of at most $(2t\tau + 1)w_l$ to the total weight mapped to j (the "+1" accounts for the weight of l itself in case that $l \in$ OPT).

Thus, the total weight mapped from the jobs of OPT to j over the jobs in P_i is at most $(2t\tau + 1)w(P_i)$, which leads to at most

$$\sum_{i=0}^{\infty}(2t\tau + 1)w(P_i) \leq \sum_{i=0}^{\infty}(2t\tau + 1)\tau^{-i}w_j = (2t\tau + 1)\frac{\tau}{\tau - 1}w_j$$

weight for all sets P_i together.

The function $f(\tau) = (2t\tau + 1)\frac{\tau}{\tau-1}$ is minimized for $\tau^* = 1 + \sqrt{1 + 1/2t}$. For $\tau = \tau^*$ (which equals the choice of τ in Algorithm 4), we then obtain that at most $f(\tau^*) = (4t\tau^* + 1)w_j$ weight can be mapped to j. Since

$$f(\tau^*) = (4t\tau^* + 1)w_j \leq f(2) = (8t + 2)w_j,$$

Algorithm 4 is $(8t + 2)$-competitive as claimed. □

Another idea to process a newly arriving job j is to check the minimum weight that has to be aborted in each single stage (separately) in order to schedule j. If the total weight of the union of all these jobs (from all stages) is less than w_j/τ, we abort them in order to schedule j. This strategy avoids running into an NP-hard problem at cost of competitiveness. The problem is now that an incoming job can be rejected although it could be scheduled such that it aborts less than $1/\tau$ of its weight. This results in a competitiveness of $\Theta(t^2)$, where t is the number of stages.

4.4. Arbitrary Processing Times

In this section, we consider the basic problem with unit weights, but we allow jobs to have different processing times. However, we demand $p_j = d_j - r_j$, i.e.,

we are in the case of interval scheduling. Our greedy approach Algorithm 1 has a much worse competitive ratio for this case. The reason is that, shortly after accepting a job j, there can appear many 'short' jobs which finish before j.

Let the longest processing time corresponding to a task in stage i be at most $\Delta(i)$ times the shortest one in stage i, i.e., $\Delta(i) := \max_{j,k} \frac{p_j^i}{p_k^i}$. If GREEDY schedules the job j with the largest processing time in stage i, OPT can schedule at most $\lceil \Delta(i) \rceil - 1$ jobs in the interval $I^\epsilon = [r_j^i + \epsilon, d_j^i - \epsilon]$ for a sufficiently small $\epsilon > 0$, plus one job k with $r_j^i < d_k^i < r_j^i + \epsilon$ and one job l with $d_j^i - \epsilon < r_l^i < d_j^i$ on each side of $I = [r_j^i, d_j^i)$ that can overlap I very little. So, in each stage i besides the first one, there can be $\lceil \Delta(i) \rceil + 1$ intersecting jobs that are mapped to the same job of GO. In the first stage, there can be just $\lceil \Delta(1) \rceil$ of such jobs (cf. the proof of Theorem 4). In total, we obtain:

Lemma 12. *Consider the basic problem with arbitrary processing times. Algorithm 1 has competitive ratio* $(\sum_{i=1}^{t} (\lceil \Delta(i) \rceil + 1) - 1)$*, where* t *denotes the number of stages and* $\Delta(i) := \max_{j,k} \frac{p_j^i}{p_k^i}$*. In particular, setting* $\Delta := \max_i \lceil \Delta(i) \rceil$*, Algorithm 1 is* $(t \cdot (\Delta + 1) - 1)$*-competitive.*

It is not obvious how to generate a lower bound for a general algorithm. However, if we take a look at arbitrary algorithms that do not abort jobs (as our greedy approach), we immediately see that those algorithms face the same problem as GREEDY: We present one large job at the beginning with processing time $\Delta >> 1$. If the algorithm does not accept the job, we do not present any further jobs and, thus, the algorithm cannot be competitive. On the other hand, if the algorithm accepts the first job, we present many jobs with small processing time 1. Thus, the algorithm can only accept job 1 whereas, in the optimal schedule, all jobs but the first one are accepted, which are Δ many (cf. Figure 7). This can be done in every stage.

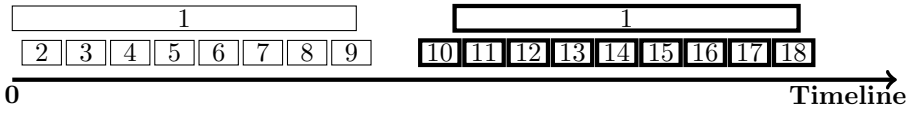

Figure 7: Example for two stages. Tasks with thick frame belong to the second stage. The tasks of the jobs two to nine are not drawn in the second stage and the tasks of the jobs ten to eighteen are not drawn in the first stage since they are all placed without any intersections. Any deterministic online algorithm that does not abort can only accept the first job whereas, in the optimal schedule, all jobs but the first one are accepted.

Lemma 13. *Let* A *be the set of deterministic online algorithms for the basic problem with arbitrary processing times that are never abort previously accepted jobs. Then, there is no algorithm in* A *with a better competitive ratio than* $\sum_{i=1}^{t} (\lceil \Delta(i) \rceil + 1) - 1$*, where* $\Delta(i) := \max_{j,k} \frac{p_j^i}{p_k^i}$*.*

Remark 14. *The one-stage approach in [19], where an already accepted job k is aborted for another job j if $r_j+p_j < r_k+p_k$, is not more promising in the case of several stages than Algorithm 1. If we abort a job in the first stage for another one that finishes earlier, this job can have tasks with very large processing times in other stages, possibly preventing the algorithm from scheduling many jobs.*

4.5. The General Model

In this section, we combine our results from the previous chapters to a more general model, i.e, there are t stages, arbitrary weights, arbitrary interval lengths, and arbitrary processing times.

Incoming jobs are scheduled as described in Algorithm 4. GREEDY is just the natural generalization of our previous algorithms. We schedule a job j as early as possible if each of its tasks fits in its respective interval (given the previous placement of the tasks of the already accepted jobs). Otherwise, we check if there is a possibility to schedule j such that less than $1/\tau$ times its weight w_j has to be aborted.

Corollary 15. *Let there be jobs with arbitrary profits that consist of tasks of arbitrary interval length in t different stages. Assume that, in each stage i, the largest processing time is at most $\Delta(i)$ times the smallest one in this stage. Then, Algorithm 4 with $\tau := 2$ is $\left(4\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)+3\right)$-competitive. In particular, with $\Delta := \max_i\lceil\Delta(i)\rceil$, it is $(4t(\Delta+1)+3)$-competitive.*

PROOF. Again, we first consider the class of algorithms that abort jobs if we can schedule another job of more than τ times their total weight instead. We map the weight of the jobs in OPT to the jobs in GO as in the proof of Theorem 11 and bound the total weight mapped to any job in GO.

Let $\Delta(i)$ be the quotient of the largest processing time and the smallest one in stage i. If GREEDY schedules a task T_j^i of job j with the largest processing time in stage i in the time interval $[x, x+p_j^i)$, it follows by the same argumentation as used in Section 4.4 that at most $\lceil\Delta(i)\rceil+1$ jobs of OPT can intersect with T_j^i and only the jobs corresponding to these tasks can map weight directly to job j in stage i. Additionally, j's own weight can be mapped to j. In total, this shows that at most $\left(\tau\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)+1\right)w_j$ weight is mapped directly to a job j scheduled by GREEDY.

Moreover, similar as before, it follows that jobs of OPT can map weight at most

$$\sum_{k=1}^{\infty} w_j\cdot\tau^{-k+1}\left(\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)\rceil)+1\right) = \frac{\tau}{\tau-1}w_j\left(\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)+1\right)$$

to j over the jobs that were sequentially aborted by j. Hence, with $\tau=2$, we obtain a competitiveness of $4\left(\sum_{i=1}^{t}(\lceil\Delta(i)\rceil+1)\right)+3$. □

4.6. Several Parallel Machines per Stage

In this section, we analyze the basic problem with several parallel machines in each stage, i.e., there is a set M_i of m_i identical parallel machines available in stage i on which the task T_j^i of a job j corresponding to stage i can be processed.

Our algorithm is again a greedy strategy, placing each task of a job on some idle machine of those available for the stage.

Algorithm 5 GREEDY for the basic problem with parallel machines

Accept a newly arriving job j if there is a machine available from M_i in every stage i on which T_j^i does not intersect with any previously accepted jobs. If several machines are available for scheduling T_j^i in some stage i, schedule T_j^i on the one with the smallest index.

In contrast to the basic problem with a single machine per stage, we have to decide on which machine out of those in M_i we place the task. For each stage i, GREEDY checks M_i for an available machine. If there is more than one idle machine, we choose the one with the smallest index. Again, we prove the competitiveness for two stages and then show how to extend this result to t stages.

Theorem 16. *Algorithm 5 is 4-competitive for the basic problem with parallel machines and two stages.*

PROOF. Let GO be the set of jobs scheduled by GREEDY and OPT be the set of jobs in an optimal schedule. We define a mapping $\phi : \text{OPT} \to \text{GO}$ and bound the number of preimages of every job in GO:

Case 1: If $j \in \text{OPT} \cap \text{GO} : \phi(j) := j$.

Case 2: If $j \in \text{OPT} \setminus \text{GO}$, job j was rejected at its arrival because, in at least one stage, it could not be scheduled on any machine. Let i_1 be the index of the machine on which OPT schedules T_j^1 in the first stage and i_2 be the index of the machine on which OPT schedules T_j^2 in the second stage.

Case 2.1: In the schedule of GREEDY, machine i_1 is busy at time r_j^1 with the task T_k^1 of some job k. Then, we set $\phi(j) := k$.

Case 2.2: In the schedule of GREEDY, machine i_1 is idle at time r_j^1, but the second task of job j does not fit on machine i_2 since there are one or two jobs blocking the machine. Of these jobs, let k be the one with the earlier release date in the second stage. Then, we set $\phi(j) := k$.

Now, let $j \in \text{GO}$. We bound the number of preimages of j. From the first case, we can get one preimage. Moreover, since we always map jobs $j' \in \text{OPT} \setminus \text{GO}$ to jobs that GREEDY schedules on the machine on which OPT

schedules j' in the corresponding stage, we get at most one preimage from Case 2.1 and at most two preimages from Case 2.2 as for the single machine problem. In total, we have $|\phi(j)^{-1}| \leq 4$.

Note that, in contrast to the single machine problem, a job $j \in \text{GO}$ can have preimages from Cases 1 and 2 simultaneously if ALG and OPT schedule job j on different machines. □

Since, again, all further stages can be treated as the second stage, we obtain the following corollary:

Corollary 17. *Algorithm 5 is $2t$-competitive for the basic problem with parallel machines and t stages.* □

Considering the same number m of parallel machines in each stage and presenting m copies of the jobs in the instance from the proof of Proposition 6, we obtain the following result:

Corollary 18. *The competitive ratio of Algorithm 5 is at least $2t - 1$ for the basic problem with parallel machines and t stages.* □

In the following, we analyze the problem variations that we considered before for the case of several parallel machines in each stage. To generalize our greedy algorithms and their analysis of to this case, we use the following argumentation: As we have seen in the proof of Theorem 16, we can define the mapping ϕ such that a job $j \in \text{OPT} \setminus \text{GO}$ assigns its weight only to jobs accepted by GREEDY that intersect with j. In particular, for each such job, at least one of its tasks has to be scheduled by GREEDY on the same machine on which OPT schedules T_j^i in some stage i.

Thus, if we denote the machines on which OPT schedules the tasks T_j^i of some job $j \in \text{OPT} \setminus \text{GO}$ by $i_1, \ldots, i_t$ (one machine for each stage), ϕ maps the weight w_j to jobs in GO of which GREEDY schedules the task of at least one stage k exactly on the machine i_k on which OPT schedules T_j^k. Thus, we just have to regard a single machine in each stage when considering the weight mapped to a job in GO and the analysis works as before.

However, as before, we define ϕ such that it always maps the weight of a job $j \in \text{OPT} \cap \text{GO}$ to itself (even if the job is scheduled on different machines by GREEDY and OPT in all stages). Therefore, a job $j \in \text{OPT} \cap \text{GO}$ mapping its weight to itself constitutes the only case where the weight of a job $j \in \text{OPT}$ can be mapped to a job in the schedule of GREEDY whose tasks GREEDY and OPT place on *different* machines in *all* stages.

For arbitrary weights, we adjust Algorithm 5 accordingly:

Algorithm 6 GREEDY for the basic problem with parallel machines and arbitrary weights

1: Accept a newly arriving job j if there is a machine available from M_i in every stage i on which T_j^i does not intersect with any previously accepted job.
2: Otherwise, compute a minimum weight set $\mathrm{Int}(j)$ of already accepted jobs whose abortion makes it possible to schedule all tasks of job j. In particular, a certain machine is selected in each stage i. Accept j aborting the jobs in $\mathrm{Int}(j)$ if the weight of job j is strictly larger than twice the total weight of the jobs in $\mathrm{Int}(j)$.

GREEDY works similar to the single machine case. If there are idle machines, GREEDY places a newly arriving job j on those. Otherwise, it computes a minimum weight set $\mathrm{Int}(j)$ of already accepted jobs whose abortion makes it possible to schedule all tasks of job j and aborts the jobs in $\mathrm{Int}(j)$ if the weight of job j is strictly larger than twice the total weight of the jobs in $\mathrm{Int}(j)$. In particular, job j is then scheduled on exactly the same machines in each stage as the jobs in $\mathrm{Int}(j)$ were.

For the analysis, we again define the set $\mathrm{GO}(z)$ to be the set of jobs already accepted and not (yet) aborted by GREEDY at time z and, for a job $j \in \mathrm{OPT}$, the set $Q \subseteq \mathrm{GO}(r_j)$ to be the set of jobs in $\mathrm{GO}(r_j)$ that intersect with j including the job j itself if $j \in \mathrm{GO}(r_j)$. We define the weight assignment exactly as in the proof of Theorem 11.

Now we can argue analogously as in the proof of Theorem 8 with the addition that a job can always map its own weight to itself if it is scheduled on different machines by OPT and GREEDY.

Thus, there can be at most $2t-1$ jobs that can directly map weight to a job k from a set P_i (defined as before). Additionally, job k can map its own weight to itself. In total, jobs in OPT can map weight at most

$$f(\tau) = \left((2t-1)\frac{\tau}{\tau-1} + \frac{1}{\tau-1} + (2t-1)\tau + 1\right) w_j$$

to a job $j \in \mathrm{GO}$. The value $f(\tau)$ is minimized for $\tau^* = 1 + \sqrt{2t/2t-1}$. With $\tau = 2$ (as in Algorithm 6), we gain the following result:

Corollary 19. *Algorithm 6 is $(8t-2)$-competitive for the basic problem with parallel machines and arbitrary weights.* □

In the case of job scheduling and unit weights, i.e., if $d_j - r_j > p_j$ is allowed but $w_j = 1$ for all j, our greedy approach tries to find an available machine in every stage and schedules the tasks of a newly arriving job as early as possible (rejecting the job if the machines are already occupied). We can define $\phi(j)$ for a job $j \in \mathrm{OPT}$ as in the proof of Theorem 9 if we just consider the machines $i_1, \ldots, i_t$ on which a job $j \in \mathrm{OPT} \setminus \mathrm{GO}$ is scheduled in Case 2. This yields:

Corollary 20. GREEDY *is $2t$-competitive for the basic problem with parallel machines and arbitrary interval length.* □

Also, the case of arbitrary processing times and the general model are transferable by using the same arguments as above. GREEDY is defined as in the single machine case, but it first tries to find an idle machine in the machine set before considering abortion of other jobs (cf. Algorithm 7).

Algorithm 7 GREEDY for the general problem with parallel machines

1: Accept a newly arriving job j if there is a machine available from M_i in every stage i on which T_j^i can be scheduled in its interval such that it does not intersect with any previously accepted jobs (scheduling it as early as possible). If several machines are available for scheduling T_j^i in some stage i, schedule T_j^i on the one with the smallest index.
2: Otherwise, compute a minimum weight set $\mathrm{Int}(j)$ of already accepted jobs whose abortion makes it possible to schedule all tasks of job j. Accept j aborting the jobs in $\mathrm{Int}(j)$ if w_j is strictly larger than τ times the total weight of the jobs in $\mathrm{Int}(j)$, scheduling all tasks of j as early as possible.

Corollary 21. *Let there be jobs with arbitrary weights that consist of tasks of arbitrary interval lengths in t stages with parallel machine set M_i in stage i. Assume that, in each stage i, the largest processing time is at most $\Delta(i)$ times the smallest one in this stage. Let $\Delta := \max_i \lceil \Delta(i) \rceil$. Then, for $\tau = 2$, Algorithm 7 is $(4t \cdot (\Delta + 1) + 3)$-competitive.* □

Lemma 22. *The lower bounds shown on the competitiveness of our specific greedy algorithms for each setting with a single machine per stage also hold in the case of several parallel machines per stage.*

PROOF. Consider the situation that there is the same number m of machines in every stage. We simply present every job in the instance that yields the lower bound for the single machine case m times. Then, everything works exactly as before. □

4.7. Further Extensions

In this section, we take a look at some further extensions of the problem. After having studied the proof techniques from the previous sections, it is quite clear how to include them into the model.

The first extension we consider is that jobs do not need to have tasks in all stages or may have several tasks per stage (each with its own specific realeas date and deadline). However, we still assume that each job has at least one task in the first stage and the information about all tasks of the job becomes available at the earliest release date of a task in the first stage. For this situation, we observe that, in the mappings constructed in the proofs of our competitiveness results, we effectively only used t as the maximum number of tasks that a job j has in order to bound the maximum number of jobs that can intersect with j. Hence, all the competitiveness results for our algorithms extend directly if we replace the number t of stages by the maximum number of tasks of a job.

Another extension is motivated by considering the stages as the nodes of a directed line graph (see Figure 8). A natural extension is then to consider an arbitrary acyclic graph (with a fixed topological sorting that corresponds to the numbering of the stages) where each job has to be processed at a subset of the nodes given by a directed path starting at node 1 (see Figure 9). We observe that, since directed paths are just special subsets of the nodes, this extension can actually be seen a special case of the extension where a job does not need to have a task in every stage.

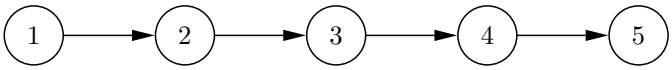

Figure 8: In our original model, the stages can be considered as a directed line graph. Here, we have five nodes that correspond to five stages.

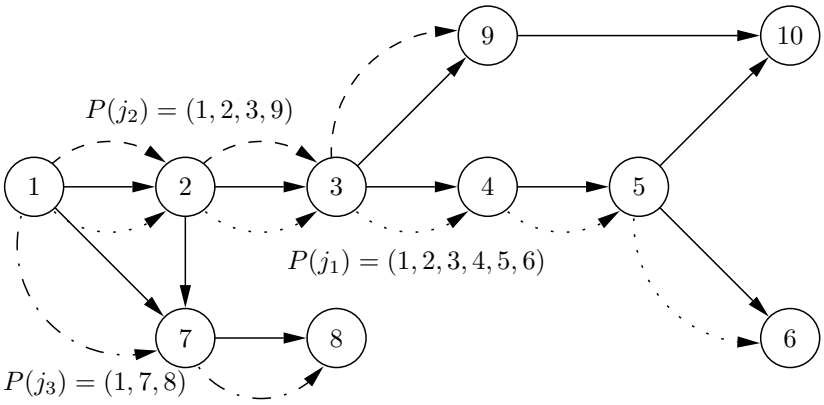

Figure 9: Scheduling jobs that correspond to directed paths in an acyclic graph corresponds to the extension where a job does not need to have a task in every stage. The node indices represent a topological sorting of the graph and correspond to the stages. The jobs are the paths on the graph indicated by the dashed and dotted arcs.

5. Experimental Results

In this section, we present experimental results on our greedy algorithms in order to get an impression of their average case behavior. We consider the basic problem, the basic problem with arbitrary weights, the basic problem with arbitrary interval lengths, and the basic problem with two parallel machines in each stage. We consider scenarios with 2, 3, 5, and 10 stages and different numbers

of arriving jobs. Each such scenario is run 100 times and our benchmark is the mean ratio OPT/ALG. The optimal solutions for the instances were calculated by solving natural integer programming formulations of the problems using Gurobi 6.0.

The first tasks of the jobs arrive uniformly distributed in the time interval $[1, 30]$ (i.e., the release dates are uniformly distributed in this interval). The tasks of each stage $s \geq 2$ arrive uniformly distributed in the time interval $[30(s-1)+1, 30s]$. We do this to ensure that the *i-th* task of a job has a release date larger than the deadline of any job of the $(i-1)$-*th* stage. In the case of arbitrary interval lengths, we uniformly distribute the release dates of each task such that no two intervals of tasks of different stages intersect, i.e., for each $i = 1, \ldots, t-1$, we introduce a gap equal to the maximum interval length between the end of the time interval in which the tasks of stage i are released and the beginning of the time interval in which the tasks of stage $i+1$ are released (for example, if the maximum interval length in stage 1 is 10, the tasks of the second stage will be released within the time interval $[40, 70]$). We can observe that the *load*, which is defined as the number of jobs released per time unit in the first stage, is of major importance for the performance of our algorithms. We regard the loads 1 (low), 2 (medium), and 5 (high), i.e., there arrive 30, 60, and 150 jobs, respectively. The weights are uniformly distributed integer values in $[1, 10000]$. The interval lengths are uniformly distributed (real) values in $[1, 10]$.

Our results are shown in Table 2. For each setting, the first line in the table contains the upper bound obtained on the competitiveness of our greedy algorithm while the other lines contain the ratios OPT/ALG observed in the computational experiments for the loads 1 (low), 2 (medium), and 5 (high). Here, the first value in each cell is the mean ratio OPT/ALG over the 100 runs, the second value is the worst ratio observed among the 100 runs. Overall, we can observe that the algorithms perform much better than their competitive ratios, especially in scenarios with a higher number of stages.

In some cases, no optimal solution could be found by Gurobi within a reasonable amount of time. For example, for some instances of the parallel machines setting with high load and two stages, the solver still produced a gap of 20% after 24 hours. Stopping the solver after ten minutes lead to an average gap of 34% for the same setting. Thus, for the instances that could not be solved optimally within ten minutes, we used the upper bound given by the Gurobi solver after ten minutes of running time to calculate the ratios shown in Table 2. Note that comparing our algorithms to the upper bound instead of the actual optimal objective value only increases the ratios observed and our algorithms would perform better when compared to the actual optimal objective values.

6. Conclusion

We studied an online flow shop scheduling problem where each job consists of several tasks that have to be completed in t different stages and the goal is to maximize the total weight of accepted jobs. We presented greedy algorithms for

Setting	Load	2 Stages	3 Stages	5 Stages	10 Stages
Basic	upper bnd	3	5	9	19
	low	1.13 (1.44)	1.18 (1.67)	1.27 (2.00)	1.32 (2.00)
	medium	1.18 (1.42)	1.26 (1.56)	1.30 (1.83)	1.39 (2.00)
	high	1.21 (1.47)	1.29 (1.64)	1.25 (1.75)	1.41 (1.80)
Arb. Weights	upper bnd	12	20	36	76
	low	1.13 (1.56)	1.18 (1.90)	1.24 (1.86)	1.34 (2.32)
	medium	1.18 (1.45)	1.26 (1.63)	1.31 (1.78)	1.45 (2.35)
	high	1.24 (1.59)	1.35 (1.86)	1.38 (1.81)	1.49 (2.21)
Arb. Int. Length	upper bnd	4	6	10	20
	low*	1.32 (1.56)	1.44 (1.75)	1.65 (2.15)	2.02 (2.50)
	medium*	1.15 (1.43)	1.28 (1.50)	1.51 (1.67)	1.92 (2.50)
	high*	1.05 (1.20)	1.16 (1.36)	1.33 (1.50)	1.65 (2.00)
Parallel Machines	upper bnd	4	6	10	20
	low	1.10 (1.29)	1.16 (1.36)	1.24 (1.64)	1.32 (1.60)
	medium	1.17 (1.33)	1.25 (1.43)	1.35 (1.67)	1.44 (1.83)
	high*	1.66 (1.84)	2.01 (2.40)	2.54 (2.86)	3.49 (4.00)

Table 2: Computational results with 100 runs for each case. The instances marked with an asterisk could not be solved optimally within a reasonable amount of time. Here, the solution produced by our algorithms was compared to the upper bound obtained by the Gurobi solver after ten minutes.

this problem and analyzed their competitiveness. Moreover, we provided lower bounds on the competitiveness achievable by any online algorithm that show that some of our algorithms actually obtain the best possible competitiveness. In addition, we showed that already the offline verison of the problem, which is a special case of the problem of finding a maximum independent set in a d-claw free graph, is APX-hard.

Directions for future research include studying further variants of the problem, e.g., by allowing preemption of tasks. Moreover, there are still gaps between some of our lower and upper bounds, which raises the question whether algorithms with a better competitive ratio than our greedy algorithms can be obtained or whether it is possible to improve the presented lower bounds. Especially for the case of several parallel machines per stage, we expect that better lower bounds can be obtained, but it is unclear how this can be achieved.

References

[1] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[2] B. Moseley, A. Dasgupta, R. Kumar, T. Sarlós, On scheduling in map-reduce and flow-shops, in: Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2011, pp. 289–298.

[3] E. M. Arkin, E. B. Silverberg, Scheduling jobs with fixed start and end times, Discrete Applied Mathematics 18 (1) (1987) 1–8.

[4] K. I. Bouzina, H. Emmons, Interval scheduling on identical machines, Journal of Global Optimization 9 (3–4) (1996) 379–393.

[5] M. C. Carlisle, E. L. Lloyd, On the k-coloring of intervals, Discrete Applied Mathematics 59 (3) (1995) 225–235.

[6] U. Faigle, W. M. Nawjin, Note on scheduling intervals online, Discrete Applied Mathematics 58 (1) (1995) 13–17.

[7] R. Canetti, S. Irani, Bounding the power of preemption in randomized scheduling, SIAM Journal on Computing 27 (4) (1998) 993–1015.

[8] G. J. Woeginger, Online scheduling of jobs with fixed start and end times, Theoretical Computer Science 130 (1) (1994) 5–16.

[9] R. Bar-Yehuda, M. M. Halldórsson, Scheduling split intervals, SIAM Journal on Computing 36 (1) (2006) 1–15.

[10] U. T. Bachmann, M. M. Halldórsson, H. Shachnai, Online selection of intervals and t-intervals, Information and Computation 233 (2013) 1–11.

[11] V. Bafna, B. O. Narayanan, R. Ravi, Non-overlapping local alignments (weighted independent sets of axis parallel rectangles), in: Proceedings of the 4th International International Workshop on Algorithms and Data Structures (WADS), 1995, pp. 506–517.

[12] P. Berman, B. DasGupta, A simple approximation algorithm for nonoverlapping local alignments (weighted independent sets of axis parallel rectangles), in: Biocomputing, Vol. 1, Springer US, 2002, pp. 129–138.

[13] M. Chlebík, J. Chlebíková, Approximation hardness of optimization problems in intersection graphs of d-dimensional boxes, in: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 267–276.

[14] B. Chandra, M. Halldórsson, Greedy local improvement and weighted set packing approximation, in: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 169–176.

[15] P. Berman, A d/2 approximation for maximum weight independent set in d-claw free graphs, Nordic Journal of Computing 7 (3) (2000) 178–184.

[16] M. M. Halldórsson, Approximating discrete collections via local improvements, in: Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 160–169.

[17] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.

[18] C. Papadimitriou, M. Yannakakis, Optimization, approximation, and complexity classes, Journal of Computer and System Sciences 43 (3) (1991) 425–440.

[19] U. Faigle, W. M. Nawjin, Greedy k-coverings of interval orders, Technical Report 979, University of Twente (1991).