INCREASING SPEED SCHEDULING AND FLOW SCHEDULING

SEBASTIAN STILLER AND ANDREAS WIESE

Abstract. Network flows and scheduling have been studied intensely, but separately. In many applications a joint optimization model for routing and scheduling is desireable. Therefore, we study flows over time with a demand split into jobs. Our objective is to minimize the weighted sum of completion times of these jobs. This is closely related to preemptive scheduling on a single machine with a processing speed increasing over time. For both, flow scheduling and increasing speed scheduling, we provide an EPTAS. Without release dates we can proof a tight approximation factor of $(\sqrt{3}+1)/2$ for Smith's rule, by fully characterizing the worst case instances. We give exact algorithms for some special cases and a dynamic program for speed functions with a constant number of speeds. We can proof a competitive ratio of 2 for the online version. We also study the class of blind algorithms, i.e., those which schedule without knowledge of the speed function. For both online, and blind algorithm we provide a lower bound.

1. Introduction

Scheduling and network flows are two corner stones of combinatorial optimization. These topics are intensely treated, and rich both in theory and applicability to real-world optimization problems. Still, many real-world applications in logistic, traffic, and telecommunication require a coupled optimization of scheduling and routing decisions. As an example consider the container terminal (cf. [5]) of a modern harbor where containers are carried from the storage area to the loading cranes by automatically guided vehicles. One has to make a scheduling decision on the order in which the containers are brought to the ships, and a routing decision for the vehicles through the small area between storage and cranes. These applications usually surpass the algorithmic means developed separately for scheduling and flows. Thus, one either has to reside to general purpose methods, in particular IP models, or to decouple the optimization of scheduling and routing. In this work we take a first step towards joint, combinatorial optimization of flows and schedules.

We consider the following, simple, but hitherto untreated model. We are given an s-t-network, static capacities for the arcs' in-flow rates, and static transit times for the arcs. Further, we are given a demand comprised of k jobs, each characterized by its own flow demand and weight. The goal is to find a feasible flow over time minimizing the sum of weighted completion times, i.e., the points in time when the complete demand of a job has reached the sink. The problem is related to earliest arrival flows and gives rise to a variation we call *multiple deadline flows*. More importantly, in contains scheduling problems belonging to a class which is interesting in its own right: *increasing speed scheduling*. Increasing speed scheduling

This work was partially supported by the DFG-research center Matheon and the DFG-focus program 1307.

appears to be untreated, and forms an interesting specialization of scheduling with varying speed and a wide generalization of scheduling with rejection. Though flow scheduling only leads to scheduling with step wise increasing speed, we also treat increasing speed scheduling for arbitrary (integrable) speed functions.

Clearly, our model is simpler than that required in the exemplary application. But this paper treats fundamental models and techniques for such problems. Towards a more realistic model we add release dates and consider different online models.

Related work. To the best of our knowledge *flow scheduling* has not been considered so far. It has some far resemblance to flow shop problems. There is a close relation to dynamic multi-commodity, earliest arrival, and universally maximal flows.

For the single source, single sink case earliest arrival flows (EAF) always exist [4], and can be found by a pseudopolynomial successive shortest path algorithms [12, 15]. There are instances where these algorithm take exponentially many steps in a binary encoded input. For multiple sinks and sources EAFs need not exist [3]. In [9] a fully polynomial-time approximation scheme for the earliest arrival s-t-problem is given. These results have been extended in [2] to solve EAFs with multiple sources.

The solution of a flow scheduling is a multicommodity flow over time where all commodities have common source and common sink, optimizing a for flows unusual objective function, namely, weighted sum of completion times. Multicommodity flow over times are NP hard even in the fractional case, and even for strongly restricted graph classes [6, 7]. See also reference therein for a survey on flows over time in general.

Increasing speed scheduling with release dates clearly contains $1||r_i, pmtn||\sum w_jC_j$ as an NP-hard special case. For this an EPTAS is known [1]. For scheduling with arbitrary varying speed, in particular, when machines stop, we can argue that the $1||pmtn||\sum w_jC_j$ problem is weakly NP-hard by a reduction from partition similar to that in [11].

Another closely related problem is scheduling with rejection (cf. [8] and references there within), i.e., jobs can be excluded from the schedule at a fixed penalty cost. This is weakly NP-hard for a single machine (reduction from knapsack). Moreover, the case of unit weights and the case of unit lengths are shown to be polynomial. The case where rejection costs are proportional to job weights is open. This is equivalent to a special case of increasing speed scheduling, notably, when the machine has constant speed until time t, and infinite (or sufficiently large) speed after t.

Definitions. This work treats the following two problems:

Definition 1 (Flow scheduling problem)**.** Consider a directed graph G with two distinct nodes s and t. For each arc we are given a static capacity, a static inflow-rate, and a static transit time. Also, we are given a set of jobs $\mathcal{J}$ where each job J_i has a weight w_i and a demand ℓ_i. The goal is to find a multi-commodity flow over time from s to t with $|\mathcal{J}|$ commodities such that $\sum_{J_i\in\mathcal{J}} w_iC_i$ is minimized where C_i denotes the time when flow value corresponding to commodity i has reached ℓ_i.

For the definition of dynamic flows over time we refer to [2, 9].

Definition 2 (Increasing speed scheduling (ISS) problem)**.** Given a machine whose speed is given as a integrable, weakly monotonically increasing function $s : \mathbb{R}^+ \to$

$\mathbb{R}^+$ and k jobs $J_i = (w_i, \ell_i)$. Compute a schedule which minimizes the weighted sum of completion times, i.e., we look for k integrable indicator functions $\chi_i : \mathbb{R}^+ \to \{0,1\}$ with $\chi_i(x) \cdot \chi_j(x) = 0$ for all $x \in \mathbb{R}^+$ and $i \neq j$ such that $C_i := \inf_{T \in \mathbb{R}^+} \{\int_0^T \chi_i(x)s(x)\mathrm{d}x \geq \ell_i\}$ exists and $\sum_{J_i \in \mathcal{J}} w_i C_i$ is minimized.

For a job (w_i, ℓ_i) we call w_i/ℓ_i its *Smith's ratio*. A schedule processing the jobs successively with increasing Smith's ratio is called a *Smith's rule algorithm*. Recall that an EPTAS is a family of $(1+\varepsilon)$-algorithms for all $\varepsilon > 0$ with running time in $O\left(f(\varepsilon) \cdot poly(n)\right)$ for a function f depending only on ε.

Our contribution. In Section 2 we establish the connection between flow scheduling and increasing speed scheduling. We show that flows that are maximal for a given set of deadlines can be found in polynomial time (Theorem 5). Next, we extend the EPTAS of [1] for preemptive single machine scheduling with release dates to the case of machines with increasing speed. Together with Theorem 5 this yields an EPTAS for the flow scheduling problem with release dates. In Section 4 we give exact, polynomial time algorithms for the ISS problem in similar special cases as considered in [8] for scheduling with rejection. Moreover, we device a dynamic program in case the speed function is a step function with constantly many steps. In Section 5 we can show that Smith's rule is a $(\sqrt{3}+1)/2$ approximation for ISS and flow scheduling. This is a tight analysis, because we achieve the result by constructively characterizing worst instances for Smith's rule. In the final sections we study online algorithms and algorithms that have no knowledge of the speed function (blind algorithms). For both cases we show a lower bound. For the online case we achieve a competitive ratio of 2. A blind algorithm with approximation factor α yields an α-approximation for the flow scheduling problem. So we get a $(\sqrt{3}+1)/2$ approximation for the flow scheduling problem.

An intriguing question that we have to leave open is, whether the ISS problem without release dates is NP-hard. Recall that it is NP-hard with release dates.

2. From Flows to Scheduling

In flow scheduling we consider flows over time with single source and sink[1]. For these it is known that earliest arrival flows (EAF) exist. Therefore it is easy to see that any such flow scheduling problem has an optimal solution with the following structure.

Let I be the smallest interval in time such that in the complement of I no flow arrives at the sink. During I the inflow rate at the sink is always positive and the interval can be partitioned into k consecutive intervals such that during each of these intervals $[T_i, T_i+1)$ all inflow to the sink belongs to the same job. Intuitively, the flow over time is a machine with varying speed. To minimize the weighted sum of completion time in the absence of release dates it is best to process the jobs without preempting in a certain order on that machine.

We can further assume, this machine has a processing speed given by the inflow rate of the EAF at the sink as a function of time. This rate is a piecewise constant function which can be calculated in pseudo-polynomial time, but it is also known to potentially have pseudo-polynomially many break-points.

[1]As we want to be brief on this please cf. [2, 9] for basic definitions and properties.

Naturally, the question arises, how to solve a flow scheduling in less than pseudo-polynomial time. Assuming that the optimal order of the jobs is known this is possible by the following concept.

Definition 3 (Multiple Deadline Flows (MDF))**.** Given an s-t-digraph G with non-negative, constant transit time τ and capacities u on the arcs, and a finite set of deadlines $\mathcal{T} = \{T_1, \dots, T_k\}$. A flow over time x for transit times τ respecting at any point in time u as arc capacities for the inflow rate is called a Multiple Deadline Flow (MDF), if for $1 \leq i \leq k$ its value up to time T_i is maximal among all feasible flows over time on (G, τ, u).

2.1. **A submodular algorithm for MDF.** An MDF can be found in polynomial time. Before we give a combinatorial algorithm for this problem, we present a reduction to the single source quickest transshipment problem.

Consider the capacitated digraph $\tilde{G} = (\tilde{V} := V \cup (\{t_1, \dots, t_{k-1}\}, \tilde{E} := E \cup \{e_i := (t, t_i) | 1 \leq i \leq k-1\}, \tilde{u})$, where u extends to $\tilde{u}$ by $\tilde{u}(e) = \infty$ for all $e \in \tilde{E} \setminus E$. We assume $\mathcal{T}$ to be sorted increasingly. Extend τ to $\tilde{\tau}$ by $\tilde{\tau}(e_i) = T_k - T_i$ for all $1 \leq i \leq k-1$. Denote $t =: t_k$ for convenience. Then compute D_i the values of maximum dynamic flows for time horizons T_i. Define $d_i := D_i - D_{i-1}$ for all $2 \leq i \leq k$ and $d_1 := D_1$ to be the demands of the terminals t_i and $-D_k$ as the demand of the source s.

Call $\tilde{T}$ the time a quickest transshipment, $\tilde{x}$, needs on the resulting network. At time $\tilde{T} - \tau(e_i)$ this optimal flow must have sent (at least) D_i units of flow through t. Else the flow could not reach all of the terminals $t_j (i \leq j)$ until time $\tilde{T}$. Therefore, this flow—restricted to the original network, $\tilde{x}|_G$—is an MDF, iff $\tilde{T} = T_k$. But $\tilde{T} \geq T_k$ follows by the same argument. An MDF on G can immediately be extended to a dynamic transshipment on $\tilde{G}$. Hence, $\tilde{T} \leq T_k$ and we have reduced MDF to quickest transshipment. A quickest transshipment can be solved in polynomial time, using an algorithm of Hoppe and Tardos [10] that minimizes a submodular function.

2.2. **A combinatorial MDF algorithm.** The submodular function in Hoppe's algorithm is a tribute to an ingredient of the general quickest transshipment problem, which our reduction does not exploit: namely that there can be multiple sinks and multiple sources. Therefore it is promising to look for a more direct algorithm, which will give us more insight to the structure of the solution. This we show next.

We keep the above notation, in particular $|\mathcal{T}| = k$. Moreover, $p_{(a,b)}$ denotes the subpath of a (simple) s-t-path p from a to b, if a and b are vertices in p, of which a is closer to s. If a or b are arcs, they denote their starting vertex. Finally, $\hat{e}$ denotes the residual arc to e.

In order to analyse the algorithm, we alternate it slightly. We often change a circle to a path and vice versa. Therefore, we write $p + e$ for the circle that arises when closing the path p by arc e.

We changed the lines 5 to 8 (and irrelevantly line 1). In the second version of the algorithm at line 5 we calculate a Min-Cost-Circulation on the original, static network, G, and in the first version on a residual network, $G_{\bar{x}}$, for some feasible, static flow, $\bar{x}$, on G. If we add to the later the circulation given by the set of circles $P + (t, s)$, where P is a path decomposition of $\bar{x}$, we will also arrive at a Min-Cost-Circulation on G. We may assume that sum of circulations to be identical to the circulation found in the second version. Thus the circulation found in the first

AlgData : $G, \mathcal{T}$
AlgResult: x
begin
1 $\bar{x} \equiv 0, P = \emptyset$;
2 Sort $\mathcal{T}$ increasing;
3 **foreach** $T \in \mathcal{T}$ **do**
4 $G' = G_{\bar{x}} \cup \{(t,s)\}, \tau((t,s)) = T_i, u((t,s)) = \infty$;
5 Compute static Min-Cost-Circulation c' on G' for cost function τ;
6 Turn c' to a static flow x' on $G_{\bar{x}}$;
7 Find standard s-t-path decomposition P' for x';
8 $P = P \cup P', \bar{x} = \bar{x} + x'$;
end
$x(e,\theta) = \sum_{\substack{e \in p \in P \\ \tau(p_{(s,e)}) \le \theta \\ \theta \le \tau(p_{(e,t)})}} |p| - \sum_{\substack{\hat{e} \in p \in P \\ \tau(p_{(s,\hat{e})}) \le \theta \\ \theta \le \tau(p_{(\hat{e},t)})}} -|p| \quad \forall e \in G, 0 \le \theta \le T_k$;
9
10 **return** x
end

Algorithm 1: MDF-Algorithm

AlgData : $G, \mathcal{T}$
AlgResult: x
begin
1 $P = \emptyset$;
2 Sort $\mathcal{T}$ increasing;
3 **foreach** $T \in \mathcal{T}$ **do**
4 $G' = G \cup \{(t,s)\}, \tau((t,s)) = T_i, u((t,s)) = \infty$;
5 Compute static Min-Cost-Circulation c' on G' for cost function τ, starting with the circulations $P + (t,s)$;
6 Turn c' to a static flow x' on G;
7 Find non-standard s-t-path decomposition P' for x', starting with the path in P;
8 $P = P'$;
end
$x(e,\theta) = \sum_{\substack{e \in p \in P \\ \tau(p_{(s,e)}) \le \theta \\ \theta \le \tau(p_{(e,t)})}} |p| - \sum_{\substack{\hat{e} \in p \in P \\ \tau(p_{(s,\hat{e})}) \le \theta \\ \theta \le \tau(p_{(\hat{e},t)})}} -|p| \quad \forall e \in G, 0 \le \theta \le T_k$;
9
10 **return** x;
end

Algorithm 2: MDF-Algorithm, alternative interpretation

algorithm at line 5 together with the (circular) flow on $P + (t,s)$ can be the same as the circulation found in the second algorithm at line 5. We assume w.l.o.g. that they are identical.

At line 7 and 8—after deleting the (t,s)-arcs—we decompose the flows on G into sets of paths, P. We will show later that the decompositions invoked are actually possible. For the moment, observe that we may obtain the same set of paths, P, in either of the two algorithms. This is true, because both by definition

contain the set of paths P', and both are decompositions that stem from the same Min-Cost-Circulation, as we have established before.

All together, we know that both algorithms on every instance may produce the same set of paths and therefore return the same dynamic flow x.

For the second algorithm it is clear from the analysis of the standard algorithm for quickest flow and the corresponding results of Hoppe and Tardos [9] on the time expansion of non-standard path decompositions, that the algorithm produces a maximum dynamic flow for the time horizon T_k. Actually, if we jumped out of the **foreach**-section after j iterations, we would have a maximum dynamic flow for time horizon T_j. By the first algorithm it is clear that the path decomposition P contains a set of paths, namely those found in the first iteration of the **foreach**-section, which gives rise to a (standard) time expansion, which is a maximum dynamic flow for time horizon T_1. An easy induction on k gives that the set P admits a partition into sets P_i, such that the (non-standard) time expansions of the sets $\bigcup_{i=1}^{j} P_i$ are maximum dynamic flows for the time horizons T_j. It is left to establish, that the paths in a later subsets of P (i.e. $P_i, i > j$) do not spoil the optimality at time T_j.

First, observe that in the jth iteration of the first algorithm the path decomposition of the flow on $G_{\bar{x}}$, say P_j, contains paths longer than T_{j-1} only. If there was a shorter path in $G_{\bar{x}}$ the circulation found in the previous iteration would not be minimal. And, as P' is decomposed into standard path on $G_{\bar{x}}$, all its path will be longer than T_{j-1}. Note that though the union of the path decompositions is a non-standard path decomposition on G, each set P_j is found as a standard path decomposition on some $G_{\bar{x}}$.

Now, consider the time expanded graph, G^{T_k}. Insert the expanded paths of P_j iteratively. In order to diminish the amount of flow that reached the sink when the expansions up to the one of P_{j-1} are already in G^{T_k} the expanded paths added in the jth step had to use an (residual) arc originating from one of the terminals t_θ, with $\theta \leq T_{j-1}$. But, as all of them have to start from a source s_θ, with $0 \leq \theta$, they need to be shorter than T_{j-1}. But this is impossible for a path in P_j, as we have just shown.

Finally to the token we left open above: We still have to proof, that the flows that occurring during the algorithm can be fully decomposed into paths, as claimed by the algorithm. In the algorithm of Ford and Fulkerson for quickest flow, this is due to the fact, that the static graph on which we look for a Min-Cost-Circulation has only one negative arc, namely (t, s). Thus all standard, negative cycles must contain this arc, and hence turn into path after its deletion. But in our case, we come across residual graphs. These may contain other negative arcs. Yet, they must not contain a negative arc before the arc (t, s) is added. Else, the negative cycle could have been used by the Min-Cost-Circulation of the previous iteration already. This contradicts its optimality.

Lemma 4. *Given a finite set of deadlines and a digraph with inflow rate capacities and transit times on the arcs. An MDF can be found in strongly polynomial time.*

From this we immediately get:

Theorem 5. *There is a strongly polynomial time algorithm for flow scheduling given that the optimal order of the jobs is known.*

3. EPTAS

In this section we present an EPTAS for the increasing speed scheduling problem with arbitrary release dates, i.e., an approximation algorithm with approximation ratio $1+\varepsilon$ and runtime $O\left(f(\varepsilon)\cdot poly(n)\right)$ for a function f depending exponentially only on ε. We show later that this yields also an EPTAS for the flow scheduling problem.

Let $\varepsilon > 0$. We describe an algorithm which guarantees an approximation ratio of $1+O(\varepsilon)$. By abuse of notation whenever we use the term $O(\varepsilon)$ we refer to a function bounded by $k\cdot\varepsilon$ for some positive k. For technical reasons, we assume that $\varepsilon < 1$. W.l.o.g. we assume that at any time the speed of the machine is at least 1. First, we derive a couple of properties which we can assume for the instance and the schedules without losing more than a factor of $1+O(\varepsilon)$ in the objective function in comparison to the optimum. Some of the techniques used here are borrowed from [1] and adapted to our problem. Then, we show how to compute the optimal schedule with these properties by a dynamic program. We will use the notions "with $1+\varepsilon$ loss" and "with $1+O(\varepsilon)$ loss", meaning that by requiring a certain property for the schedule we might lose at most a factor of $1+\varepsilon$ or $1+O(\varepsilon)$, respectively, in the optimal objective function value. In each of the following lemmata we assume that all adjustments allowed by virtue of all previous lemmata have already been done.

We define $R(w)$ to be the timestep when the total work that the machine has done so far equals w. As short notation we use $R_x := R\left((1+\varepsilon)^x\right)$. Note that since we assume that the machine always runs with at least unit speed we have that $R_{x+1} \leq (1+\varepsilon)\, R_x$. We split the time scale into intervals of the form $I_x := [R_x, R_{x+1})$. In order to simplify notation we will use the notion I_x for the interval as well as for the work that the machine does within I_x. Note that $I_x = \varepsilon\,(1+\varepsilon)^x$.

Lemma 6. *With $1+O(\varepsilon)$ loss, we can assume for each job J_i that $r_i \geq R\left(\varepsilon\ell_i\right)$ and that ℓ_i is a power of $(1+\varepsilon)$.*

Proof. Assume we are given an optimal schedule OPT. We construct a new schedule OPT' as follows: whenever a job J_i is processed within a time interval $[x, y]$ in OPT then it is processed in the interval $[(1+\varepsilon)\, x, (1+\varepsilon)\, y]$ in OPT'. We say we *scale time* by a factor of $1+\varepsilon$. We have that $OPT' \leq (1+\varepsilon)\, OPT$. The resulting schedule is feasible for an instance in which J_i has a processing demand of $(1+\varepsilon)\,\ell_i$. Thus, we can safely shift the release time of J_i to $\max\left\{r_i, R\left(\varepsilon\ell_i\right)\right\}$ and still obtain a valid schedule for processing demand ℓ_i.

Scaling time again by a factor of $1+\varepsilon$ again yields that a demand of $(1+\varepsilon)\,\ell_i$ is processed for each job J_i. We increase the demand of J_i to the largest value $(1+\varepsilon)^x$ (for an integer x) such that $(1+\varepsilon)^x \leq (1+\varepsilon)\,\ell_i$. □

Lemma 7. *Assume the adjustments of Lemma 6. With $1+\varepsilon$ loss, we can additionally assume that each job is released at a time R_x.*

Proof. In this proof we use a technique which we call *interval-hopping*. Consider any schedule S. For each interval I_x, we take the work that is done in I_x in S and process it in the interval I_{x+1} rather than in I_x. This results in a loss of at most $1+\varepsilon$. Now each job J_i which was originally released within an interval $I_x := [R_x, R_{x+1})$ is not processed before R_{x+1}. Thus, we can safely move its release time to R_{x+1}. □

In order to simplify the complexity of the problem we want that to calculate the objective function as if each job finished at the end of an interval. Therefore, for the remainder of this section we do not consider the objective function $\sum_{J_j \in \mathcal{J}} w_j C_j$ but the objective function $\sum_{J_j \in \mathcal{J}} w_j \min \{R_x : C_j \leq R_x\}$. As an effect, we can assume that the machine has constant speed within an interval.

Lemma 8. *Assume the adjustments of the previous lemmata. For completion times C_i resulting from any schedule we have that $\sum_{J_i \in \mathcal{J}} w_i C_i \leq \sum_{J_i \in \mathcal{J}} w_i \min \{R_x : C_i \leq R_x\} \leq (1+\varepsilon) \sum_{J_i \in \mathcal{J}} w_i C_i$.*

Proof. The first inequality is obvious. The second inequality holds since $\min \{R_x : C_i \leq R_x\} \leq (1+\varepsilon)\, C_i$ for all values C_i. □

In the algorithm we will distinguish between *large* and *small jobs*. A job J_i is small if it is released at a time R_x such that $\ell_i \leq \varepsilon I_x$. Otherwise it is large. We denote by H_x and T_x the large and small jobs, respectively, which were released at time R_x. We introduce the following lemma in order to show that there is a $(1+\varepsilon)$-approximate schedule which has a simple structure.

Lemma 9. *Assume the adjustments of the previous lemmata. With $1 + O(\varepsilon)$ loss, we can additionally assume that*

- *no small job is ever preempted,*
- *no small job is processed in more than one interval,*
- *the order in which the small jobs are executed obeys Smith's rule,*
- *each large job $J_i \in H_x$ is preempted only if there is an integer $k \leq \frac{1}{\varepsilon^3}$ such that a fraction of exactly $k \cdot \frac{\varepsilon I_x}{\ell_i}$ of the job has already been processed, and*
- *at any point in time in each set H_x there is at most one job which has already been processed but which has not been finished yet.*

Proof. We can assume the last claim without any loss. For the other claims we again use the technique of interval-hopping. For the sake of analysis, we first consider a relaxation of our instance I (i.e., an instance I' such that $OPT(I') \leq OPT(I)$). Starting with $OPT(I')$ we construct a schedule S for I such that $S \leq (1+\varepsilon)^2\, OPT\,(I')$.

The instance I' is defined as follows: Let $\delta > 0$ be a constant which divides the demands of all small jobs. Then we replace each small job J_i by $\frac{\ell_i}{\delta}$ jobs with demand δ and weight $\delta w_i/\ell_i$, all with the same release date as J_i. We call those new jobs the *tiny jobs*. Clearly, in this instance it is optimal to schedule the small jobs according to Smith's rule, i.e., whenever a small job is scheduled the available job with the highest Smith's ratio is scheduled. W.l.o.g. we can assume that if in $OPT\,(I')$ a tiny job is processed which corresponds to a small job J_i then for all other small jobs $J_{i'}$ we have that either all of their tiny jobs are already scheduled or none of them (i.e., the tiny jobs corresponding to the different small jobs do not mix). Also, w.l.o.g. we assume that in $OPT\,(I')$ jobs are only preempted at the end of intervals (since jobs are released only at the beginning of intervals).

We now perform an interval-hop of two intervals: For a loss of $(1+\varepsilon)^2$ we take the work that is done in each interval I_x in the schedule $OPT\,(I')$ and process it in the interval I_{x+2} rather than in I_x. We call the resulting schedule $OPT\,(I')_{hop}$. However, now in each interval I_{x+2} (which processes jobs which were processed in I_x by $OPT\,(I')$) there is a spare space of at least $2\varepsilon I_x$. We have that

$OPT(I')_{hop} \leq (1+\varepsilon)^2 OPT(I')$. We define the instances I_{hop} and I'_{hop} by taking I and I', respectively, and for each small/tiny job which was released at time R_x we move its release time to R_{x+2}. Note that $OPT(I')_{hop}$ is a valid schedule for I'_{hop} in which the tiny jobs are ordered according to Smith's rule.

Now we construct a schedule S for I_{hop} based on $OPT(I')_{hop}$ which has the properties stated in the lemma. First, we process each small job J_i whenever its corresponding tiny jobs were processed within $OPT(I')_{hop}$. Now there can be at most one small job J_i which starts within one interval I_{x+2} and does not finish within it. If there is such a job J_i then we use at most εI_x of the spare space to finish J_i. If at the end of I_{x+2} a large job $J_i \in H_y$ is preempted then - using again at most εI_x of the spare space - we can continue processing it until a fraction of exactly $k \cdot \frac{\varepsilon I_y}{\ell_i}$ of the job has been processed for an integer k. Due to our slightly changed objective function we have that $S \leq OPT(I')_{hop} \leq (1+\varepsilon)^2 OPT(I') \leq (1+\varepsilon)^2 OPT(I)$. □

For a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ we denote by $p(\mathcal{J}')$ their total demand.

Lemma 10. *Assume the adjustments of the previous lemmata. Without any loss we can assume that*

- *the number of distinct job sizes in H_x is bounded by $|H_x| \leq 3\log_{1+\varepsilon}\frac{1}{\varepsilon} + 1$,*
- *the number of jobs in each distinct size is bounded by $1/\varepsilon$, and*
- *$p(T_x) \leq I_x$.*

Proof. Let $J_i \in H_x$. Since J_i is large we know that $\ell_i > \varepsilon I_x$. Due to Lemma 6 we have that $r_i = R_x \geq R(\varepsilon \cdot \ell_i)$ which implies that $(1+\varepsilon)^x \geq \varepsilon \cdot \ell_i$. Since all demands are powers of $1+\varepsilon$ the number of distinct job sizes equals the number of integers y such that

$$\begin{array}{rccccc} & \varepsilon^2(1+\varepsilon)^x & < & (1+\varepsilon)^y & \leq & \frac{1}{\varepsilon}(1+\varepsilon)^x \\ \Leftrightarrow & 2\log_{1+\varepsilon}\varepsilon + x & < & y & \leq & \log_{1+\varepsilon}\frac{1}{\varepsilon} + x \\ \Leftrightarrow & 2\log_{1+\varepsilon}\varepsilon & < & y - x & \leq & \log_{1+\varepsilon}\frac{1}{\varepsilon} \end{array}$$

This implies that $|H_x| \leq 3\log_{1+\varepsilon}\frac{1}{\varepsilon} + 1$. At most $\frac{I_x}{\varepsilon I_x} = 1/\varepsilon$ jobs of each particular size can be scheduled in I_x. Within each size the jobs are ordered by their weight. Thus, we can safely move the release times of all other jobs of each size to R_{x+1}. Note that here we need that $\varepsilon < 1$ since this is necessary for $\varepsilon^2(1+\varepsilon)^x < \frac{1}{\varepsilon}(1+\varepsilon)^x$.

For the small jobs recall that due to Lemma 9 we can assume that they are ordered by Smith's rule. Also, we assume that no small job is processed in more than one interval. Thus, we can assume that $p(T_x) \leq I_x$: We sort the jobs in T_x non-decreasingly by their Smith's ratio and then pick jobs according to this order until the next job would not fit in I_x anymore. The release time of all other jobs can safely be moved to R_{x+1} since we will not process them in R_x anyway. □

The following lemma gives an upper bound on the time a job has to wait before it completes.

Lemma 11. *Assume the adjustments of the previous lemmata. With $1+\varepsilon$ loss, we can additionally assume that each job which is released at time R_x finishes in the interval $I_{x+s(\varepsilon)}$ the latest, where $s(\varepsilon)$ is a constant which does only depend on ε.*

Proof. We use interval-hopping again and shift the work being done in each interval I_x to the interval I_{x+1}. From Lemma 10 we conclude that $p(T_x) + p(H_x) \leq$

$I_x + \frac{1}{\varepsilon} \cdot \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1\right) \cdot \frac{1}{\varepsilon} (1+\varepsilon)^x$. We define

$$s(\varepsilon) := \left\lceil \log_{1+\varepsilon} \left(\frac{1}{\varepsilon} + \frac{1}{\varepsilon^4} \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1 \right) \right) \right\rceil + 1$$

Our interval-hop creates a spare space of size εI_x in each interval I_{x+1}. Thus, in the interval $I_{x+s(\varepsilon)}$ there is now a spare space of at least $\varepsilon I_{x+s(\varepsilon)-1}$. We calculate that

$$\begin{aligned} I_x + \frac{1}{\varepsilon} \cdot \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1 \right) \cdot \frac{1}{\varepsilon} (1+\varepsilon)^x &= (1+\varepsilon)^x \left(\varepsilon + \frac{1}{\varepsilon^2} \cdot \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1 \right) \right) \\ &= (1+\varepsilon)^x \cdot \varepsilon^2 \cdot \left(\frac{1}{\varepsilon} + \frac{1}{\varepsilon^4} \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1 \right) \right) \\ &\leq (1+\varepsilon)^x \cdot \varepsilon \cdot I_{s(\varepsilon)-1} \\ &= \varepsilon \cdot I_{x+s(\varepsilon)-1} \end{aligned}$$

and thus we can process all jobs $T_x \cup H_x$ in the spare space in the interval $I_{x+s(\varepsilon)}$. This implies that there is a $(1+\varepsilon)$-approximative solution in which all jobs $T_x \cup H_x$ finish in the interval $I_{x+s(\varepsilon)}$ the latest. □

Now we partition the ordered list of the jobs in each set T_x into at most $2/\varepsilon^2$ *packs,* each with size at most $\varepsilon^2 \cdot I_x$. Denote by $P_{x,i}$ the ith pack of small jobs which are released at time R_x.

Lemma 12. *Assume the adjustments of the previous lemmata. With $1+\varepsilon$ loss, we can additionally assume that*

- *in each interval I_x either all or none of the jobs in a pack $P_{x',i}$ are scheduled,*
- *each job which is released at time R_x finishes in the interval $I_{x+s(\varepsilon)+2}$ the latest, and*
- *the ordering of the small jobs does not necessarily obey Smith's rule anymore.*

Proof. We use interval-hopping and shift the work which is done in each interval I_x to the interval I_{x+2}. This gives us a free space of εI_{x+1} in each interval I_{x+2}. Now consider all packs $P_{x',i}$ such that some but not all of the jobs in $P_{x',i}$ are scheduled within I_x. Due to the original ordering by Smith's Rule, this holds for at most one pack from each release time. The total demand of these packs is upper bounded by

$$\begin{aligned} \sum_{i=x-s+1}^{x} \varepsilon^2 \cdot I_i &= \sum_{i=x-s+1}^{x} \varepsilon^3 \cdot (1+\varepsilon)^i \\ &\leq \varepsilon^3 \cdot \sum_{i=0}^{x} (1+\varepsilon)^i \\ &= \varepsilon^3 \cdot \frac{1-(1+\varepsilon)^{x+1}}{1-(1+\varepsilon)} \\ &= \varepsilon^2 \cdot \left((1+\varepsilon)^{x+1} - 1 \right) \\ &\leq \varepsilon^2 \cdot (1+\varepsilon)^{x+1} \\ &= \varepsilon \cdot I_{x+1} \end{aligned}$$

Thus, in the gained free space we can schedule all jobs from packs $P_{x',i}$ which have partly but not fully been processed. □

Before we describe the dynamic program we summarize our adjustments on the instance:

- for each job J_i we have that r_j and ℓ_j are powers of $(1+\varepsilon)$,
- for each job J_i we have it holds that $r_i \geq R(\varepsilon \ell_i)$,
- the number of jobs in H_x is bounded by a constant for each release time R_x, and
- $p(T_x) \leq I_x$.

We showed that all these adjustments lose at most a factor of $1 + O(\varepsilon)$ in the optimal objective value. Further, we showed that for an adjusted instance as above there is a $(1+O(\varepsilon))$-approximative solution with the following properties:

- no small job is ever preempted,
- the small jobs are grouped into packs and no pack is processed in more than one interval,
- each large job $J_i \in H_x$ is preempted only if there is an integer $k \leq \frac{1}{\varepsilon^3}$ such that a fraction of exactly $k \cdot \frac{\varepsilon I_x}{\ell_i}$ of the job has already been processed,
- at any point in time in each set H_x there is at most one job which has already been processed but which has not finished yet,
- the value of the solution is measured according to the objective function $\sum_{J_j \in \mathcal{J}} w_j \min\{R_x : C_j \leq R_x\}$ (which results in a higher value than the original objective function), and
- each job with release time R_x finishes before time $R_{x+s(\varepsilon)+1}$.

Now we describe the dynamic program which finds the best solution with the above properties. Each table entry is identified by a combination of

- an interval I_x,
- for each interval I_y with $x - s(\varepsilon) \leq y < x$,
  - the subset of jobs in H_y which have already been fully processed,
  - a job $J_i \in H_y$ and an integer $k \leq \frac{1}{\varepsilon^3}$ such that a fraction of exactly $k \cdot \frac{\varepsilon I_x}{\ell_i}$ of J_i has been processed, and
  - the subset of the packs $P_{y,i}$ which have already been fully processed.

Since we need to consider at most $s(\varepsilon) \cdot |\mathcal{J}|$ intervals in total, the number of table entries is bounded by

$$\begin{aligned} & (s(\varepsilon)+2) \cdot |\mathcal{J}| \cdot \left(2^{\frac{1}{\varepsilon} \cdot 3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + \frac{1}{\varepsilon}} \cdot \left(\frac{1}{\varepsilon} \cdot 3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + \frac{1}{\varepsilon}\right) \cdot \frac{1}{\varepsilon^3} \cdot 2^{\frac{2}{\varepsilon^2}}\right)^{s(\varepsilon)+2} \\ \in \quad & O\left(|\mathcal{J}| \, 2^{poly(1/\varepsilon)}\right) \end{aligned}$$

In order to compute the value for each table entry which corresponds to an interval I_x we need to enumerate all possibilities to schedule the available large jobs and packs of small jobs in I_x. Note that due to our changed objective function it does not matter at what exact time within the interval a job finishes: each job which finishes within I_x is charged R_{x+1}. For one table entry this computation can be done in time $O\left(2^{s(\varepsilon) \cdot \left(\frac{1}{\varepsilon} 3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + \frac{1}{\varepsilon} + \frac{2}{\varepsilon^2}\right)}\right)$.

The preprocessing of the jobs can be done in $O(n \cdot \log n + s(\varepsilon) \cdot n)$: It requires to

- round the release times and demands of all jobs: $O(n)$,
- partition the jobs into small and large jobs: $O(n)$,
- define the packs of small jobs and adjust their release times. This requires $O(n \log n + s(\varepsilon) \cdot n)$ since we need to sort the small jobs and define the sets T_x for each interval I_x, and
- define the adjusted release times of the large jobs. This requires $O(n \log n + s(\varepsilon) \cdot n)$ again since we need to sort the jobs and define the sets H_x for each interval I_x.

We obtain the following theorem:

Theorem 13. *There is a polynomial time approximation scheme for the increasing speed scheduling problem with release dates with running time* $O\left(2^{poly(1/\varepsilon)}n + n \log n\right)$.

In order to do the computation in the dynamic program it is not necessary to know the exact speed function. It is sufficient to know the points in time when a total demand of $(1+\varepsilon)^x$ has already been processed for the relevant values of x. Recall that at most $s(\varepsilon) \cdot |\mathcal{J}|$ intervals are relevant for us. Thus, we obtain the following theorem:

Corollary 14. *There is an efficient polynomial time approximation scheme for the flow scheduling problem with release dates.*

Proof. Follows from Theorems 5 and 13. □

4. Tractable Cases of ISS

In this section we analyze the structure of the increasing speed scheduling problem. We identify some properties which allow efficient algorithms for certain special cases. Moreover, we provide the necessary insights for our analysis of the Smith's Rule algorithm in Section 5. Throughout this section we assume that all jobs are released at time $t = 0$. Accordingly, we can restrict ourselves to non-preemptive schedules.

If all jobs have unit weight a simple exchange argument shows that it is optimal to order the jobs ascendingly by demand. However, we can prove a slightly more general result:

Theorem 15. *If in the instance there is an ordering* $J_1, J_2, ..., J_{|\mathcal{J}|}$ *for the jobs such that* $\frac{w_i}{\ell_i} \geq \frac{w_{i+1}}{\ell_{i+1}}$ *and* $\ell_i \leq \ell_{i+1}$ *for each* i *then it is optimal to order the jobs ascendingly by demand (or descendingly by ratios* $\frac{w_i}{\ell_i}$*, respectively).*

The theorem can be shown using the following lemma repeatedly.

Lemma 16. *Assume that in a schedule there are two jobs* J_i, J_j *with* $\frac{w_i}{\ell_i} \leq \frac{w_j}{\ell_j}$ *and* $\ell_i \geq \ell_j$ *and* J_j *is executed directly after* J_i*. Then the objective value does not increase if we swap* J_i *and* J_j*. If additionally* $\frac{w_i}{\ell_i} < \frac{w_j}{\ell_j}$ *then swapping* J_i *and* J_j *will strictly decrease the objective function.*

Proof. We denote the original schedule by $\langle i, j \rangle$ and the schedule obtained by swapping J_i and J_j by $\langle j, i \rangle$. Denote by t_0 the time when J_i starts in $\langle i, j \rangle$, by t_1 the time when J_j terminates in $\langle j, i \rangle$, by t_2 the time when J_i terminates in $\langle i, j \rangle$ and by t_3 the time when J_j terminates in $\langle i, j \rangle$. W.l.o.g. we assume that the machine has constant speed within the interval $[t_i, t_{i+1})$ for each $i \in \{0, 1, 2\}$ (see Figure 4.1).

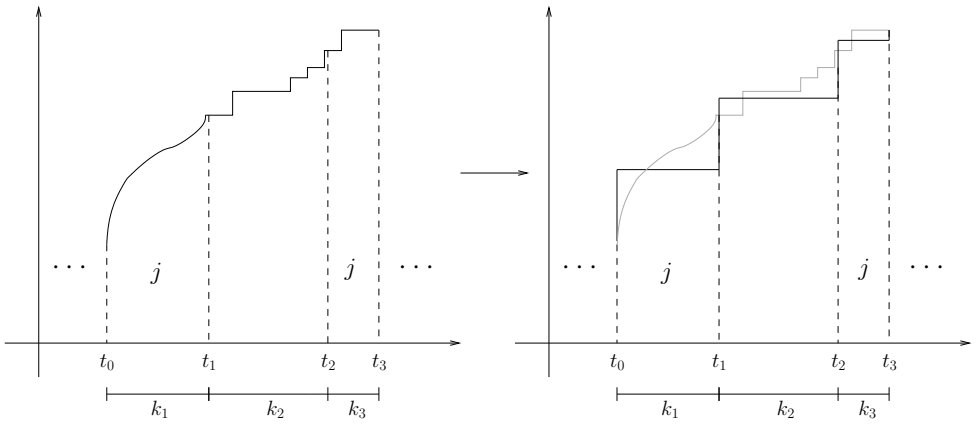

FIGURE 4.1. Proof of Lemma 16: We can assume w.l.o.g. that our machine has three different speeds within the interval $[t_0, t_3)$.

Denote the respective speeds by s_1, s_2, s_3. W.l.o.g. we assume that $s_3 = 1$. For ease of notations we define $k_i := t_i - t_{i-1}$ for $i \in \{1, 2, 3\}$.

We calculate that

$$\begin{array}{rcc} & cost\left(\langle i,j\rangle\right) - cost\left(\langle j,i\rangle\right) & \geq & 0 \\ \Leftrightarrow & t_2 \cdot w_i + t_3 \cdot w_j - t_1 \cdot w_j - t_3 \cdot w_i & \geq & 0 \\ \Leftrightarrow & t_2 \cdot w_i + t_3 \cdot w_j & \geq & t_1 \cdot w_j + t_3 \cdot w_i \\ \Leftrightarrow & (k_1 + k_2)\, w_i + (k_1 + k_2 + k_3)\, w_j & \geq & k_1 \cdot w_j + (k_1 + k_2 + k_3)\, w_i \\ \Leftrightarrow & (k_2 + k_3)\, w_j & \geq & k_3 \cdot w_i \\ \Leftrightarrow & \frac{w_j}{\ell_j} & \geq & \frac{w_i}{k_2 + \ell_j} \end{array}$$

The latter holds since $\frac{w_j}{\ell_j} \geq \frac{w_i}{\ell_i} = \frac{w_i}{\ell_j + s_2 \cdot k_2} \geq \frac{w_i}{\ell_j + k_2}$. If additionally $\frac{w_i}{\ell_i} < \frac{w_j}{\ell_j}$ then the above calculation gives strict inequality. □

For machines with constant speed one there is a well known exchange argument showing that in an optimal schedule the jobs are ordered non-decreasingly by their Smith's factors [14]. In our setting, this argument can easily be applied to jobs starting and finishing within an interval A in which the machines has constant speed. We show that the statement also holds for the set of all jobs which end in such an interval A (and not necessarily start in A).

We split the time axis into intervals in which the speed function does not change its value. We denote by $s_1, s_2, ..., s_k$ the different speeds of the machine and by $A_1, A_2, ..., A_k$ the corresponding intervals, i.e., $A_i := s\left(s_i\right)^{-1}$. Assume a schedule S is given. We say a job j is *in an interval* A_i if the finishing time of j in S lies within A_i. Denote by $\mathcal{J}_i$ the jobs which lie in the interval A_i.

Proposition 17. *In an optimal schedule the jobs in $\mathcal{J}_i$ are ordered according to Smith's Rule.*

Proof. This can be shown by an exchange argument. Assume on the contrary that there is an optimal schedule and for two jobs $J_j, J_{j'} \in \mathcal{J}_i$ we have that J_j is scheduled before $J_{j'}$ but $\frac{w_j}{\ell_j} < \frac{w_{j'}}{\ell_{j'}}$. W.l.o.g. we can assume that $J_{j'}$ is scheduled directly after J_j. We call this schedule $\langle j, j'\rangle$ and compare it with the schedule

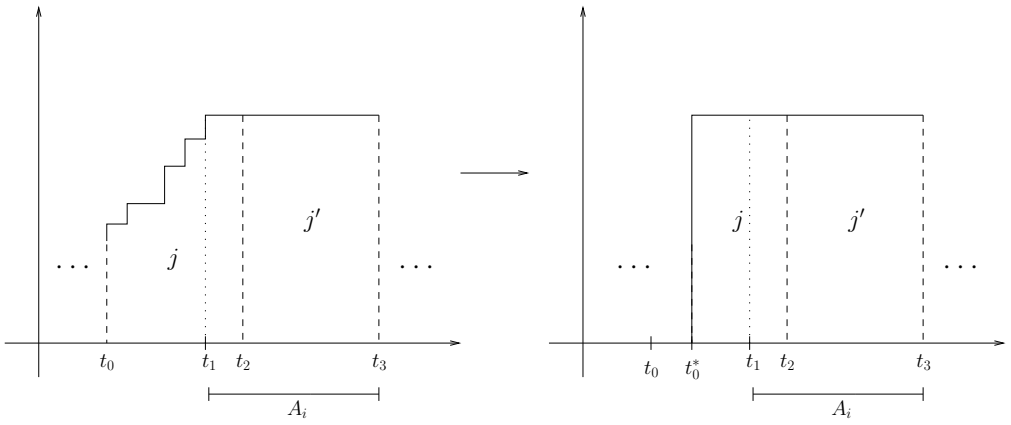

FIGURE 4.2. We can assume w.l.o.g. that our machine has three different speeds within the interval $[t_0, t_3)$.

$\langle j', j\rangle$ which is obtained by exchanging J_j and $J_{j'}$. If $\ell_{j'} \leq \ell_j$ then Lemma 16 proves the claim. So now assume that $\ell_{j'} > \ell_j$.

We denote by t_0 the starting time of J_j in $\langle j, j'\rangle$, by t_1 the starting time of interval A_i, by t_2 the finishing time of J_j in $\langle j, j'\rangle$, and finally by t_3 the finishing time of $J_{j'}$ in $\langle j, j'\rangle$. See Figure 4.2 for a sketch. We observe that the finishing times of J_j and $J_{j'}$ in both schedules ($\langle j, j'\rangle$ and $\langle j', j\rangle$) do not change if we adjust the machine as follows: Denote by t_0^* the timestep with $\int_{t_0}^{t_1} s(x)dx = (t_0^* - t_0)\, s_i$. We redefine our machines by setting $s(x) := 0$ for $x \in [t_0, t_0^*)$ and $s(x) := s_i$ for $x \in [t_0^*, t_1)$. Now in both schedules J_j and $J_{j'}$ start and finish within interval A_i. Thus, the claim can be shown with the same argument which shows that Smith's Rule is optimal for $1||\sum w_j C_j$:

$$\begin{array}{rcc} & \langle j', j\rangle & > & \langle j, j'\rangle \\ \Leftrightarrow & w_{j'}\frac{\ell_{j'}}{s_i} + w_j\frac{\ell_j+\ell_{j'}}{s_i} & > & w_j\frac{\ell_j}{s_i} + w_{j'}\frac{\ell_j+\ell_{j'}}{s_i} \\ \Leftrightarrow & w_j \cdot \ell_{j'} & > & w_{j'} \cdot \ell_j \\ \Leftrightarrow & \frac{w_j}{\ell_j} & > & \frac{w_{j'}}{\ell_{j'}} \end{array}$$

This is contradicts our assumption that $\frac{w_j}{\ell_j} < \frac{w_{j'}}{\ell_{j'}}$. □

Knowing these properties we can present the dynamic program for the special case of limited number of speed changes. The dynamic program gets a list of the jobs ordered by Smith's Rule. It successively removes a job J_j from the list and chooses the interval $A_i = [a_i, a_{i+1})$ in which J_j finishes. Inside A_i, the job J_j is scheduled right after the last job finishing within A_i. If J_j is the first job assigned to A_i we try all start offsets less or equal a_i for which J_j finishes within A_i. Thus, in the dynamic programming table we need to encode how many jobs have already been removed from the list and how much space (at the beginning and at the end) of each interval is already occupied by jobs.

Now we describe the dynamic program in detail. First, we order the jobs in a list according to Smith's Rule. Ties are broken arbitrarily. We define $L := \sum_j \ell_j$. Note that there are at most L possible start times for a job. Each entry in the dynamic programming table is identified by

- the number of jobs which are still in the job list, i.e., an entry equal to k means that the last k jobs in the job list are still unscheduled and
- for each interval $A_i = [a_i, a_{i+1})$ there are values $t_i \leq L$ and $t'_i \leq L$ such that the intervals $\left[a_i, a_i + \frac{t_i}{s_i}\right)$ and $\left[a_{i+1} - \frac{t'_i}{s_i}, a_{i+1}\right)$ are already used by some jobs.

We store in each table entry the best possible objective value for the remaining jobs which is possible with the given constraints. Note that the number of entries in the table is bounded by $n \cdot L^{2k}$. The value for an entry $(k, t_1, t'_1, ..., t_k, t'_k)$ is computed as follows: We take the job J_{n-k+1} and try to schedule it. We try each interval A_i: if $t_i > 0$ we try to schedule J_{n-k+1} with start time $a_i + \frac{t_i}{s_i}$. If $t_i = 0$ we try to schedule it with each start time $a_i - \frac{m}{s_i}$ with $m < \ell_{n-k+1}$. By "try" we mean that we check carefully if scheduling the job at the respective position contradict the fact that other intervals are already occupied by some jobs. We pick the position for J_{n-k+1} which yields the minimum total objective value.

Theorem 18. *If the number of different values for the speed function is bounded by a constant there is a pseudopolynomial dynamic program which solves the ISS problem optimally.*

Proof. Computing an entry in the dynamic programming table can be done in pseudopolynomial time. The number of entries in the dynamic programming table is bounded by $n \cdot L^{2k}$. Lemma 17 implies that our procedure finds the optimal solution. □

Note that this pseudopolynomial algorithm cannot be combined with the pseudopolynomial algorithm for earliest arrival flows [12, 15] to achieve an exact, pseudopolynomial algorithm for the entire problem. An EAF corresponds to a machine with pseudopolynomially many speeds. Our result requires a *constant* number of speeds.

Now we study the special cases where all jobs have the same demand or the same Smith's factors. We will benefit from these insights in the analysis in Sections 5 and 7, respectively. The following lemma holds not only for increasing speed functions but also for speed functions which might increase or decrease.

Lemma 19. *If all jobs have the same demand then it is optimal to order the jobs descendingly by their weight. This is still true if the speed of the machine can decrease and increase.*

Proof. This can be shown by an exchange argument. Assume on the contrary that there is an optimal schedule in which a job J_i is scheduled before a job J_j but $w_i < w_j$. W.l.o.g. we can assume that J_j is scheduled directly after J_i. Then the objective value strictly decreases if we swap J_i and J_j. This contradicts that the original schedule was optimal. □

Proposition 20. *If in an instance I all jobs have the same Smith's ratio then*

- *there is an optimal schedule which orders the jobs ascendingly by demand and*
- *there is a worst possible schedule which orders the jobs descendingly by demand.*

Proof. Follows from Lemma 16 and its proof. □

5. A Tight Analysis of Smith's Rule

In this section we present a tight analysis which shows that the Smith's Rule Algorithm is exactly a $\frac{\sqrt{3}+1}{2}$-approximation. Let $I = (\mathcal{J}, M)$ be an instance of the increasing speed scheduling problem. We denote by $SR(I)$ the worst possible schedule which obeys Smith's rule (i.e., tie-breaking decisions are taken such that the total weight of the schedule is maximized). We show how to transform I into an instance with a special structure without decreasing $SR(I)/OPT(I)$. Then we show that on instances with this structure the Smith's Rule algorithm is exactly a $\frac{\sqrt{3}+1}{2}$-approximation.

Assume on the contrary that there is an instance I such that $\frac{SR(I)}{OPT(I)} > \frac{\sqrt{3}+1}{2}$, again with $SR(I)$ being the objective value of the *worst* possible schedule which obeys Smith's rule. We will show later by perturbing the weights of the jobs that one cannot prove a better approximation factor for the Smith's Rule algorithm even if all tie breaking decisions are done optimally.

In the following lemmas we show how one can transform I into an instance with a special structure without decreasing $SR(I)/OPT(I)$. First we define a partition of the jobs in I into classes $\mathcal{C}_i$ such that two jobs belong to the same class if and only if they have the same Smith's ratio. The classes are sorted descendingly by the Smith's ratios of their elements, i.e., for two jobs $J_i \in \mathcal{C}_{i'}$ and $J_j \in \mathcal{C}_{j'}$ it holds that $\frac{w_i}{\ell_i} \geq \frac{w_j}{\ell_j}$ if and only if $i' \leq j'$. Now we show that there are instances with the worst possible fraction $SR(I)/OPT(I)$ in which all jobs have a Smith's ratio of 1.

Lemma 21. *For any instance I there is an instance $I' = (\mathcal{J}', M)$ such that $w_i/\ell_i = 1$ for all jobs $J'_i \in \mathcal{J}'$ and $\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$. Moreover, if in I the demands of all jobs are integral then in I' the demands of all jobs are integral as well.*

Proof. Let $\tilde{I}$ be an instance with as few classes $\mathcal{C}_i$ as possible such that $\frac{SR(\tilde{I})}{OPT(\tilde{I})} \geq \frac{SR(I)}{OPT(I)} =: \alpha$. If in $\tilde{I}$ there is only one class then we can scale the weights of the jobs such that $w_i/\ell_i = 1$ for all jobs and we are done. So now assume that there are at least two classes in $\tilde{I}$. Denote by $SR(\mathcal{C}_i)$ and $OPT(\mathcal{C}_i)$ the amount that a class $\mathcal{C}_i$ contributes $SR\left(\tilde{I}\right)$ and $OPT\left(\tilde{I}\right)$, respectively. Since $\frac{SR(\tilde{I})}{OPT(\tilde{I})} \geq \alpha$ there must be a class $\mathcal{C}_k$ such that $\frac{SR(\mathcal{C}_k)}{OPT(\mathcal{C}_k)} \geq \alpha$. Now there are two cases:

(1) The class $\mathcal{C}_k$ is the class with the highest Smith's ratio (i.e., $k = 1$). We create I' by removing the jobs of all other classes from $\tilde{I}$. We have that $SR(I') = SR(\mathcal{C}_k)$ and $OPT(I') \leq OPT(\mathcal{C}_k)$ since in $SR(\tilde{I})$ the jobs in $\mathcal{C}_k$ are the first jobs which are scheduled. This implies that $\frac{SR(I')}{OPT(I')} \geq \frac{SR(\mathcal{C}_k)}{OPT(\mathcal{C}_k)} \geq \alpha$.

(2) The class $\mathcal{C}_k$ is not the class with the highest Smith's ratio (i.e., $\mathcal{C}_k \neq \mathcal{C}_1$). Then we increase the weights of the jobs in $\mathcal{C}_k$ until they all have the Smith's ratio of $\mathcal{C}_{k-1}$. Denote by I' the resulting instance. Clearly, I' has one class less than I. Let $\beta > 1$ denote the factor by which we increased the weights of the jobs in $\mathcal{C}_k$. Then we calculate that $OPT(I') \leq OPT\left(\tilde{I}\right) + (\beta - 1)\, OPT(\mathcal{C}_k)$ and $SR(I') = SR(\tilde{I}) + (\beta - 1)\, SR(\mathcal{C}_k)$. Then we have

that

$$\begin{aligned}\frac{SR(I')}{OPT(I')} &\geq \frac{SR(\tilde{I}) + (\beta-1)\,SR\left(\mathcal{C}_k\right)}{OPT\left(\tilde{I}\right) + (\beta-1)\,OPT\left(\mathcal{C}_k\right)} \\ &\geq \frac{\alpha\left(OPT\left(\tilde{I}\right) + (\beta-1)\,OPT\left(\mathcal{C}_k\right)\right)}{OPT\left(\tilde{I}\right) + (\beta-1)\,OPT\left(\mathcal{C}_k\right)} \\ &= \frac{SR(I)}{OPT(I)}\end{aligned}$$

Note that in both cases above we do not introduce any fractional job demands. Thus, if in I the demands of all jobs are integral then in I' the demands of all jobs are integral, too. □

Recall from Proposition 20 that if all Smith's ratios are identical then we can assume that $OPT(I)$ orders the jobs ascendingly by demand and the worst Smith's Rule schedule $SR(I)$ orders the jobs descendingly by demand. Now we want to study the demands of the jobs. W.l.o.g. we assume that all jobs have integral demands. Assume that in $\mathcal{J} = \left\{J_1, J_2, ..., J_{|\mathcal{J}|}\right\}$ the jobs are ordered ascendingly by their demand. Let k denote the largest integer such that $\sum_{i=1}^{k} \ell_i \leq \frac{1}{2}\sum_{i=1}^{|\mathcal{J}|} \ell_i$. We define $\mathcal{J}_{small} := \{J_1, J_2, ..., J_k\} \subset \mathcal{J}$. For an instance I' we denote the respective set by $\mathcal{J}'_{small}$.

Lemma 22. *For any instance $I = (\mathcal{J}, M)$ such that all jobs in $\mathcal{J}$ have a Smith's ratio of 1 there is an instance $I' = (\mathcal{J}', M)$ such that*

- *$\mathcal{J}'_{small}$ consist only of jobs of demand 1 and*
- *$\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$.*

Moreover, if in I the demands of all jobs are integral then in I' the demands of all jobs are integral as well.

Proof. W.l.o.g. we assume that $SR(I)$ orders the jobs increasingly by their demand and $OPT(I)$ orders the jobs decreasingly by their demand. If in $\mathcal{J}_{small}$ there are only jobs of demand 1 then there is nothing to show. So now assume that $\mathcal{J}_{small}$ contains a job whose demand is at least two. Let J_j denote the job with smallest index such that $\ell_j \geq 2$.

For $I' = (\mathcal{J}', M)$ we use the machine M without any changes and we define $\mathcal{J}' := \mathcal{J}\backslash\{J_j\}\cup\{J_{j'}, J_{j''}\}$ where $J_{j'}$ is a job with demand $\ell_{j'} := 1$ and $J_{j''}$ is a job of demand $\ell_{j''} := \ell_j - 1$. Since we want the Smith's factors of all jobs to be 1 we define $w_{j'} := 1$ and $w_{j''} := \ell_j - 1$. Note that $J_1, J_2, ..., J_{j-1}, J_{j'}, J_{j''}, J_{j+1}, ..., J_{|\mathcal{J}|}$ orders the jobs in $\mathcal{J}'$ in increasing order. Figure 5.1 shows a sketch of the modification. Now we show that $\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$.

Denote by t_1 the start time of J_i in $OPT(I)$. Denote by t_2 and t_3 the finish time of $J_{j'}$ and $J_{j''}$ in $OPT(I')$, respectively. Similarly, let $\bar{t}_1$ be the start time of J_j in $SR(I)$. Denote by $\bar{t}_2$ the finish time of $J_{j''}$ in $SR(I')$ and by $\bar{t}_3$ the finish time of $J_{j''}$ in $SR(I')$. Figure 5.1 shows a sketch. We define $SR_\Delta := SR(I) - SR(I') = \bar{t}_3 \cdot \ell_j - \bar{t}_3 \cdot \ell_{j'} - \bar{t}_2 \cdot \ell_{j''}$ and $OPT_\Delta := OPT(I) - OPT(I') = t_3 \cdot \ell_j - t_2 \cdot \ell_{j'} - t_3 \cdot \ell_{j''}$. In order to show that $\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$ we first prove that $SR_\Delta \leq OPT_\Delta$ (in other words: the optimal solution saves more than the Smiths rule solution when

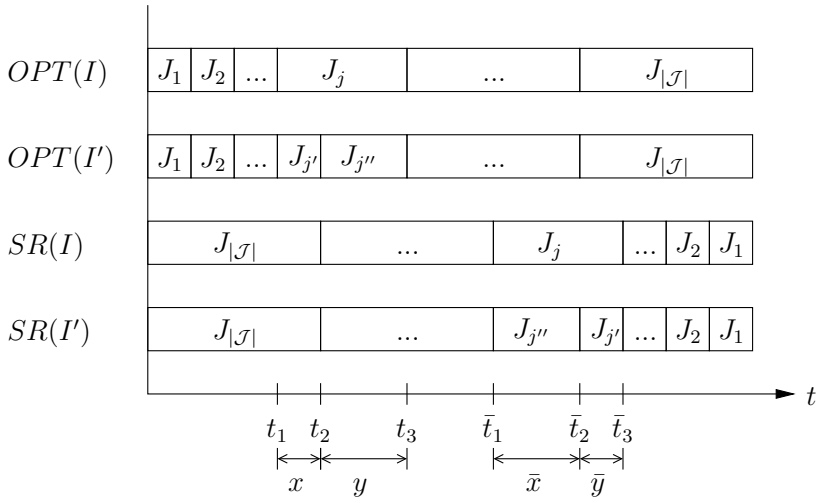

FIGURE 5.1. The sketch shows the jobs in I and I' in the orders in which they are sorted in SR and OPT. To simplify the readability of the sketch we assume that the machine runs constantly with unit speed.

we replace J_j by $J_{j'}$ and $J_{j''}$). W.l.o.g. we assume that the machine M runs constantly with speed s in the time interval $[t_2, t_3)$. Similarly, we assume that M runs with speed $\bar{s}$ in the time interval $[\bar{t}_2, \bar{t}_3)$. Note that $s \leq \bar{s}$ since $J_j \in \mathcal{J}_{small}$ and thus $t_3 \leq \bar{t}_2$. We define $x := t_2 - t_1$ and $y := t_3 - t_2$. Similarly, we define $\bar{x} := \bar{t}_2 - \bar{t}_1$ and $\bar{y} := \bar{t}_3 - \bar{t}_2$.

We calculate that

$$\begin{aligned} OPT_\Delta &= (t_1 + x + y)\,\ell_j - (t_1 + x)\,\ell_{j'} - (t_1 + x + y)\,\ell_{j''} \\ &= y\ell_j - y\ell_{j''} \end{aligned}$$

and

$$\begin{aligned} SR_\Delta &= (\bar{t}_1 + \bar{x} + \bar{y})\,\ell_j - [(\bar{t}_1 + \bar{x})\,\ell_{j''} + (\bar{t}_1 + \bar{x} + \bar{y})\,\ell_{j'}] \\ &= \bar{y}\ell_j - \bar{y}\ell_{j'} \end{aligned}$$

and thus

$$\begin{aligned} OPT_\Delta - SR_\Delta &= y\ell_j - y\ell_{j''} - \bar{y}\ell_j + \bar{y}\ell_{j'} \\ &= (\ell_j - \ell_{j''})\left(\frac{\ell_j - 1}{s}\right) + \frac{1}{\bar{s}}\,(1 - \ell_j) \\ &\geq (\ell_j - \ell_{j''})\left(\frac{\ell_j - 1}{\bar{s}}\right) + \frac{1}{\bar{s}}\,(1 - \ell_j) \\ &= \frac{1}{\bar{s}}\,[(\ell_j - \ell_{j''})\,(\ell_j - 1) + 1 - \ell_j] \\ &= 0 \end{aligned}$$

We define $\beta := \frac{OPT(I)-OPT_\Delta}{OPT(I)}$ and $\gamma := \frac{SR(I)-OPT_\Delta}{SR(I)}$. A simple calculation shows that $\gamma \geq \beta$. We further obtain that

$$\begin{aligned}
\frac{SR(I')}{OPT(I')} &= \frac{SR(I) - SR_\Delta}{OPT(I) - OPT_\Delta} \\
&\geq \frac{SR(I) - OPT_\Delta}{OPT(I) - OPT_\Delta} \\
&= \frac{\gamma \cdot SR(I)}{\beta \cdot OPT(I)} \\
&\geq \frac{\beta \cdot SR(I)}{\beta \cdot OPT(I)} \\
&\geq \frac{SR(I)}{OPT(I)}
\end{aligned}$$

We repeat the operation described above until we obtain an instance in which there are only jobs of demand 1 in the respective set $\mathcal{J}_{small}$. □

Now we are interested in the first job which does not have demand 1. Denote this job by J_m. Let s_m and t_m denote its start and finish times in $OPT(I)$ and let $\bar{s}_m$ and $\bar{t}_m$ denote its start and finish times in $SR(I)$.

Lemma 23. *For any instance I such that all jobs in $\mathcal{J}$ have a Smith's ratio of 1, $\mathcal{J}_{small}$ contains only jobs of demand 1, and the demands of all jobs are integral, there is an instance $I' = (\mathcal{J}', M)$ such that*

- *$\mathcal{J}'_{small}$ consist only of jobs of demand 1,*
- *$\bar{s}_m \leq s_m$,*
- *$\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$, and*
- *all jobs in $\mathcal{J}'$ have integral demand and a Smith's ratio of 1.*

Proof. If $\bar{s}_m > s_m$ then we apply in J_m the procedure described in the proof of Lemma 22. This is possible since the start times of J_m guarantee that $\bar{s} \geq s$ (with the notation in the proof). □

Now we show how we can change our instance I to an instance I' in which there is at most one (long) job which is not contained in $\mathcal{J}_{small}$.

Lemma 24. *Let $\varepsilon > 0$. For any instance I such that all jobs in $\mathcal{J}$ have a Smith's ratio of 1, $\mathcal{J}_{small}$ contains only jobs of demand 1, and $\bar{s}_m \leq s_m$ there is an instance $I' = (\mathcal{J}', M')$ such that*

- *$\mathcal{J}'_{small}$* consist only of jobs of demand 1,
- $|\mathcal{J}' \setminus \mathcal{J}'_{small}| = 1$ (i.e., there is exactly one job which is larger than 1),
- M' has at most one speed change which occurs when the large job is finished in $SR(I')$, and
- $\frac{SR(I')}{OPT(I')} \geq (1-\varepsilon)\,\frac{SR(I)}{OPT(I)}$.

Proof. First, we do the following two changes on I in order to obtain an instance I'': we combine all jobs which are not contained in $\mathcal{J}_{small}$ to one long job $J_{m''}$. Then we dramatically increase the speed of M after a timestep t^* to be defined later.

Denote by s_i and t_i the start and finish time of each job in $OPT(I)$ and by $\bar{s}_i$ and $\bar{t}_i$ the start and finish time of each job in $SR(I)$. Assume that $J_1, J_2, ..., J_{|\mathcal{J}|}$

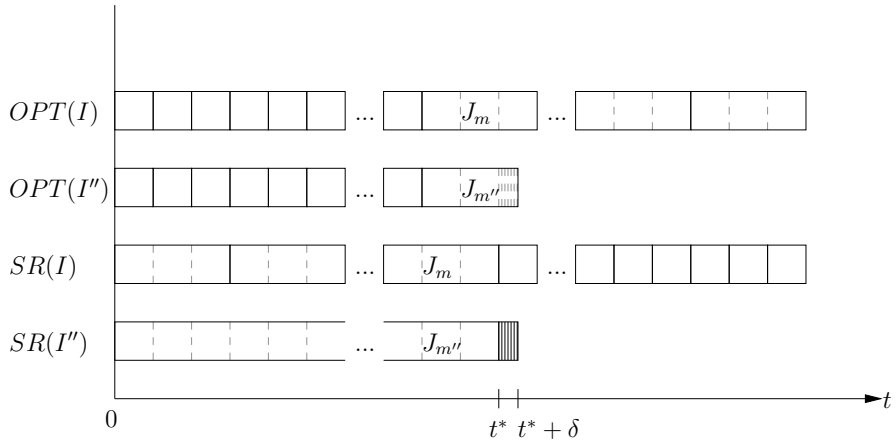

FIGURE 5.2. The time intervals in which the jobs are processed according to the respective schedules. In order to simplify the sketch we assume that M runs constantly with unit speed. It is easy to see that M'' runs significantly faster than M after t^*.

orders the jobs in $\mathcal{J}$ ascendingly by demand. We define $I'' = (\mathcal{J}'', M'')$ as follows: $\mathcal{J}'' := \mathcal{J}_{small} \cup \{J_{m''}\}$ where $J_{m''}$ is a new job with weight $w_{m''} := \ell_{m''} := \sum_{i=m}^{|\mathcal{J}|} \ell_i$.

Moreover, we obtain M'' with the following operations: We start with M. Let t^* denote the timestep with $\int_0^{t^*} s(x)dx = \ell_{m''}$. Let $\delta > 0$ be a constant to be defined later. In the time interval $[t^*, t^* + \delta]$ we define M'' to have speed $|\mathcal{J}''_{small}|/\delta$. We allow only values for δ such that M'' never slows down (i.e., there are not timesteps x and x' with $x < x'$ but $s''(x) > s''(x')$ with s'' being the speed function of M''). Note that M'' completes all jobs within the interval $[0, t^* + \delta]$.

We claim that for any given $\varepsilon > 0$ there is a $\delta > 0$ such that $\frac{SR(I'')}{OPT(I'')} \geq (1 - \varepsilon) \frac{SR(I)}{OPT(I)}$.

In order to analyze $\frac{SR(I'')}{OPT(I'')}$ we split the jobs into *bricks* of demand 1. Let J_i be a job. For J_i we introduce ℓ_i bricks $B_{i,1}, ..., B_{i,\ell_i}$. Let $pay_{SR(I)}(J_i)$ the amount that J_i contributed towards the objective function in $SR(I)$. We define $pay_{SR(I)}(B_{i,j}) := pay_{SR(I)}(J_i)/\ell_i$ for all j with $1 \leq j \leq \ell_i$. We define $pay_{OPT(I)}(J_i)$ and $pay_{OPT(I)}(B_{i,j})$ similarly. Let $\mathcal{B}$ denote the set of all bricks. Note that some of the bricks correspond to J''_m in I'' and to some other jobs from $\mathcal{J} \setminus \mathcal{J}_{small}$ in I. We say a brick $B_{i,j}$ is *processed* in $OPT(I)$ during a time interval $[s, t]$ if until time s a fraction of $\frac{j-1}{\ell_i}$ of J_i is processed in $OPT(I)$ and until time t a fraction of $\frac{j}{\ell_i}$ of J_i is processed in $OPT(I)$.

For a schedule S we define $\mathcal{B}_{early}(S)$ to be all bricks which are processed up to time t^* in S. We define $\mathcal{B}_{late}(S) := \mathcal{B} \setminus \mathcal{B}_{early}(S)$. We observe that for all bricks $B_{i,j} \in \mathcal{B}_{early}(OPT(I))$ we have that $pay_{OPT(I)}(B_{i,j}) \geq pay_{OPT(I'')}(B_{i,j})$. Moreover, for all bricks $B_{i,j} \in \mathcal{B}_{late}(OPT(I))$ we have that $pay_{OPT(I)}(B_{i,j}) \leq t^* + \delta$. We observe that for all bricks $B_{i,j} \in \mathcal{B}_{early}(SR(I))$ we have that $pay_{SR(I'')}(B_{i,j}) \geq pay_{SR(I)}(B_{i,j})$. Moreover, for all bricks $B_{i,j} \in \mathcal{B}_{late}(SR(I))$ we have that $pay_{SR(I'')}(B_{i,j}) \geq t^*$. Figure 5.2 shows a sketch of the above.

We calculate that

$$\begin{aligned}\frac{SR(I'')}{OPT(I'')} &= \frac{\sum_{B\in\mathcal{B}_{early}(SR(I''))} pay_{SR(I'')}(B) + \sum_{B\in\mathcal{B}_{late}(SR(I''))} pay_{SR(I'')}(B)}{\sum_{B\in\mathcal{B}_{early}(OPT(I''))} pay_{OPT(I'')}(B) + \sum_{B\in\mathcal{B}_{late}(OPT(I''))} pay_{OPT(I'')}(B)} \\ &\geq \frac{\sum_{B\in\mathcal{B}_{early}(SR(I))} pay_{SR(I)}(B) + |\mathcal{B}_{late}(SR(I))|\cdot t^*}{\sum_{B\in\mathcal{B}_{early}(OPT(I))} pay_{OPT(I)}(B) + |\mathcal{B}_{late}(OPT(I))|\cdot (t^*+\delta)}\end{aligned}$$

We observe that $\sum_{B\in\mathcal{B}_{late}} pay_{OPT(I)}(B) \geq \sum_{B\in\mathcal{B}_{late}} pay_{SR(I)}(B) \geq |\mathcal{B}_{late}(SR(I))|\cdot t^* = |\mathcal{B}_{late}(SR(I))|\cdot t^*$. This implies that

$$\frac{\sum_{B\in\mathcal{B}_{early}(SR(I))} pay_{SR(I)}(B) + |\mathcal{B}_{late}(SR(I))|\cdot t^*}{\sum_{B\in\mathcal{B}_{early}(OPT(I))} pay_{OPT(I)}(B) + |\mathcal{B}_{late}(OPT(I))|\cdot t^*} \geq \frac{SR(I)}{OPT(I)}$$

(in other words: if we increased the speed of M to infinity after time t^* then the optimal schedule saves more than the Smith's rule schedule). Thus, for every given $\varepsilon > 0$ there is a $\delta > 0$ such that

$$\frac{\sum_{B\in\mathcal{B}_{early}(SR(I))} pay_{SR(I)}(B) + |\mathcal{B}_{late}(SR(I))|\cdot t^*}{\sum_{B\in\mathcal{B}_{early}(OPT(I))} pay_{OPT(I)}(B) + |\mathcal{B}_{late}(OPT(I))|\cdot (t^*+\delta)} \geq (1-\varepsilon)\frac{SR(I)}{OPT(I)}$$

It remains to modify I' such that the machine has at most two different speeds (without reducing the factor between the two schedules). Recall that in $\mathcal{J}''$ there are some jobs of demand 1 and one large job. Moreover, in $OPT(I'')$ the jobs of demand 1 are all finished before t^*. We define $I' := (\mathcal{J}'', M')$ and specify M' as follows: we start with M''. Let $z := \int_0^{t^*} s'(x)dx/t^*$ (i.e., z is the average speed of the machine during the time interval $[0, t^*]$. Then we define

- $s''(x) := z$ for all $x \in [0, t^*)$ and
- $s''(x) := s'(x)$ for all $x \in [t^*, t^*+\delta]$

We have that $SR(I') = SR(I'')$ and $OPT(I') \leq OPT(I'')$. The former holds since the Smith's rule schedule cannot benefit from our adjustments. The latter holds since at each point in time M' has processed at least as much as M'' (i.e., the finishing times of the jobs cannot increase). □

Now we prove the approximation ratio of $\frac{\sqrt{3}+1}{2}$ for the algorithm.

Theorem 25. *For any instance I it holds that $\frac{SR(I)}{OPT(I)} \leq \frac{\sqrt{3}+1}{2}$.*

Proof. Let $\varepsilon > 0$. Using the lemmata above we derive an instance $I' = (\mathcal{J}', M')$ with the following properties:

- for all jobs $J'_i \in \mathcal{J}'$ it holds that $w'_i/\ell'_i = 1$,
- the demands (and thus the weights) of all jobs are integral,
- in $SR(I')$ the jobs are sorted descendingly by their demand,
- in $OPT(I')$ the jobs are sorted ascendingly by their demand, and
- J'_{small} consists only of jobs of demand 1.
- There is exactly one job in $\mathcal{J}'$ which does not have demand 1. Denote by ℓ'_m its demand.
- M' has at most one speed change which occurs when the large job is finished in $SR(I')$. Denote by t^* the time of the speed change.
- $\frac{SR(I')}{OPT(I')} \geq (1-\varepsilon)\frac{SR(I)}{OPT(I)}$.

We define $L := \sum_{J'_i \in \mathcal{J}'} \ell'_i$. Let s_1 and s_2 denote the two speed values of M' with $s_1 \leq s_2$. Denote by t_{max} the time when the last job finishes. We set $k := L - \ell'_m$ (i.e., we have k small jobs of demand 1) and calculate that

$$\begin{aligned} SR(I') &= t^* \cdot \ell'_m + \sum_{i=1}^{k} \left(t^* + \frac{i}{s_2} \right) \\ &= t^* \cdot \ell'_m + k \cdot t^* + \frac{k\,(k+1)}{2 \cdot s_2} \\ &= L \cdot t^* + \frac{k\,(k+1)}{2 \cdot s_2} \end{aligned}$$

and

$$\begin{aligned} OPT(I') &= \ell'_m \cdot t_{max} + \sum_{i=1}^{k} \frac{i}{s_1} \\ &= \ell'_m \cdot t_{max} + \frac{k\,(k+1)}{2 \cdot s_1} \end{aligned}$$

We substitute $t_{max} = \frac{\ell'_m}{s_1} + \frac{k}{s_2}$ and obtain

$$\begin{aligned} \frac{SR(I')}{OPT(I')} &= \frac{\frac{\ell'_m}{s_1} \cdot k + \frac{\left(\ell'_m\right)^2}{s_1} + \frac{k(k+1)}{2s_2}}{\frac{(\ell'_m)^2}{s_1} + \frac{k \cdot \ell'_m}{s_2} + \frac{k(k+1)}{2s_1}} \\ &\leq \frac{\frac{\ell'_m}{s_1} \cdot k + \frac{\left(\ell'_m\right)^2}{s_1}}{\frac{(\ell'_m)^2}{s_1} + \frac{k(k+1)}{2s_1}} \\ &= \frac{\ell'_m \cdot k + \left(\ell'_m\right)^2}{\left(\ell'_m\right)^2 + \frac{k^2+k}{2}} \\ &\leq \frac{\ell'_m \cdot k + \left(\ell'_m\right)^2}{\left(\ell'_m\right)^2 + \frac{k^2}{2}} \end{aligned}$$

using that $k \leq \ell'_m$. For fixed ℓ'_m we define $f(k) := \frac{\ell'_m \cdot k + \left(\ell'_m\right)^2}{(\ell'_m)^2 + \frac{k^2}{2}}$. We calculate that

$$\begin{aligned} f'(k) = 0 &\Leftrightarrow \left(\ell'_m\right)^2 + \frac{k^2}{2} - k^2 - k \cdot \ell'_m = 0 \\ &\Leftrightarrow k^2 + 2k \cdot \ell'_m - 2 \cdot \left(\ell'_m\right)^2 = 0 \\ &\Leftrightarrow k = -\ell'_m \pm \sqrt{3\left(\ell'_m\right)^2} \\ &\Leftrightarrow k = \left(-1 \pm \sqrt{3}\right) \cdot \ell'_m \end{aligned}$$

and conclude with further calculus that f has its maximum for $[0, \ell'_m]$ in $k = \left(-1 + \sqrt{3}\right) \cdot \ell'_m$. We further calculate that

$$\begin{aligned}\frac{SR(I')}{OPT(I')} &\leq f\left(\left(\sqrt{3}-1\right)\cdot \ell'_m\right)\\ &= \frac{\ell'_m\cdot\left(\sqrt{3}-1\right)\cdot\ell'_m+\left(\ell'_m\right)^2}{\left(\ell'_m\right)^2+\frac{\left(4-2\sqrt{3}\right)\cdot\left(\ell'_m\right)^2}{2}}\\ &= \frac{\sqrt{3}}{1+2-\sqrt{3}}\\ &= \frac{\sqrt{3}+1}{2}\end{aligned}$$

So for any $\varepsilon > 0$ we can show that $\frac{SR(I)}{OPT(I)} \leq \frac{1}{1-\varepsilon}\cdot\frac{SR(I')}{OPT(I')} \leq \frac{1}{1-\varepsilon}\cdot\frac{\sqrt{3}+1}{2}$. This implies that $\frac{SR(I)}{OPT(I)} \leq \frac{\sqrt{3}+1}{2}$. □

Proposition 26. *For any $\varepsilon > 0$ there are instances I_ε such that $\frac{SR(I_\varepsilon)}{OPT(I_\varepsilon)} \geq \frac{\sqrt{3}+1}{2}\cdot(1-\varepsilon)$.*

Proof. Let $\varepsilon > 0$. Let ℓ_m be an integer to be defined later. We introduce $k := \left\lfloor\left(\sqrt{3}-1\right)\ell_m\right\rfloor$ jobs with demand and weight 1 and one job with demand and weight ℓ_m. We define our machine M_ε to have speed 1 in the time interval $[0,\ell_m]$ and speed s in the time interval $\left[\ell_m,\ell_m+\frac{k}{s}\right]$ (we will define s later). We calculate that

$$OPT\left(I_\varepsilon\right) \leq \frac{k(k+1)}{2}+\ell_m\cdot\left(\ell_m+\frac{k}{s}\right)$$

and

$$SR\left(I_\varepsilon\right) = \left(\ell_m\right)^2+k\cdot\ell_m+\frac{k(k+1)}{2s}$$

We further calculate that

$$\begin{aligned}\frac{SR\left(I_\varepsilon\right)}{OPT\left(I_\varepsilon\right)} &\geq \frac{\left(\ell_m\right)^2+k\cdot\ell_m+\frac{k(k+1)}{2s}}{\frac{k(k+1)}{2}+\ell_m\cdot\left(\ell_m+\frac{k}{s}\right)}\\ &= \frac{\left(\ell_m\right)^2+\left\lfloor\left(\sqrt{3}-1\right)\ell_m\right\rfloor\cdot\ell_m+\frac{\left\lfloor\left(\sqrt{3}-1\right)\ell_m\right\rfloor\left(\left\lfloor\left(\sqrt{3}-1\right)\ell_m\right\rfloor+1\right)}{2s}}{\frac{\left\lfloor\left(\sqrt{3}-1\right)\ell_m\right\rfloor\left(\left\lfloor\left(\sqrt{3}-1\right)\ell_m\right\rfloor+1\right)}{2}+\ell_m\cdot\left(\ell_m+\frac{\left\lfloor\left(\sqrt{3}-1\right)\ell_m\right\rfloor}{s}\right)}\end{aligned}$$

Now we choose s and ℓ_m large enough such that

$$\begin{aligned}\frac{SR\left(I_\varepsilon\right)}{OPT\left(I_\varepsilon\right)} &\geq (1-\varepsilon)\frac{\left(\ell_m\right)^2+\left(\left(\sqrt{3}-1\right)\ell_m\right)\cdot\ell_m}{\frac{\left(\left(\sqrt{3}-1\right)\ell_m\right)^2}{2}+\left(\ell_m\right)^2}\\ &= (1-\varepsilon)\frac{\sqrt{3}}{2-\sqrt{3}+1}\\ &= (1-\varepsilon)\frac{\sqrt{3}+1}{2}\end{aligned}$$

□

Corollary 27. *Any algorithm respecting Smith's rule is a* $\frac{\sqrt{3}+1}{2}$*-approximation, and none of these algorithms can achieve a better approximation factor for all instances.*

Proof. So far we considered the worst among these algorithms. Now consider a best among these. This algorithm respects Smith's rule, but breaks ties optimally for each instance. Let I be an instance. For any $\varepsilon > 0$ we can find a perturbation of the weights of the jobs in I yielding an instance I_ε such that

- there is only one possible ordering for the jobs in I_ε which obeys Smith's rule and still
- $\frac{SR(I_\varepsilon)}{OPT(I_\varepsilon)} \geq (1-\varepsilon)\,\frac{SR(I)}{OPT(I)}$.

Applying this reasoning to the worst-case instances presented in Proposition 26 shows that no Smith's rule algorithm can achieve a better approximation factor than $\frac{\sqrt{3}+1}{2}$.

□

6. Blind algorithms

We note that the Smith's Rule algorithm orders the jobs without knowledge of the machine. We call algorithms with this property *blind* algorithms. With Theorem 5 any polynomial time blind algorithm for the ISS problem yields a polynomial time algorithm for the flow scheduling problem. Therefore, we have the following corollary.

Corollary 28. *There is an approximation algorithm for the flow scheduling problem with approximation factor* $\frac{\sqrt{3}+1}{2}$ *which runs in polynomial time.*

Now we establish a lower bound for the possible performance ratio of blind algorithms.

Theorem 29. *The performance ratio of a blind algorithm is at best* 1.1215*.*

Proof. We consider the following instance I: We have $n+1$ jobs with $w_0 = \beta \cdot n$, $\ell_0 = n$, $w_1 = w_2 = ... = w_n = 1 = \ell_1 = \ell_2 = ... = \ell_n$ (for n and β to be defined later). A blind algorithm must schedule these jobs in some order without knowledge of the machine. We denote by S_k the schedule in which the job J_0 is scheduled at position k (note that $\{S_1, ..., S_n\}$ is the set of all possible schedules for I without preemption). We use the notation $S_{k,M}$ for the objective value obtained when executing the schedule S_k on machine M. We claim that for each schedule S_k there is a machine M such that $\frac{S_{k,M}}{OPT_M} \geq 1.1215$ (where OPT_M denotes the best possible schedule for I on M).

Let M be a machine which runs constantly with unit speed. Clearly, on M the schedule S_1 is the optimal schedule (since S_1 orders the jobs according to Smith's rule and M has no speed changes). We calculate that

$$\begin{aligned} S_{k,M} &= \sum_{i=1}^{k-1} i + (k-1+n)\cdot\beta n + \sum_{i=k}^{n} (n+i) \\ &= n^2\,(1.5+\beta) + n\,(1.5-k+\beta k-\beta) \end{aligned}$$

Since $OPT_M = S_{1,M}$ we obtain that

$$\begin{aligned}\frac{S_{k,M}}{OPT} &= \frac{S_{k,M}}{S_{1,M}} \\ &= \frac{n^2(1.5+\beta)+n(1.5-k+\beta k-\beta)}{n^2(1.5+\beta)+0.5n} \\ &=: f(n,k,\beta)\end{aligned}$$

Now for each schedule S_k we define a machine M_k by the following speed function:

$$s_k(t) = \begin{cases} 1 & \text{if } t < k+n-1 \\ L & \text{if } t \geq k+n-1 \end{cases}$$

for a large constant L to be defined later. We calculate that

$$\begin{aligned}S_{k,M_k} &\geq \sum_{i=1}^{k-1} i + \beta n(k+n-1) + (n-k+1)(k+n-1) \\ &= \frac{(k-1)k}{2} + \beta nk + \beta n^2 - \beta n + nk + n^2 - n - k^2 - kn + k + k + n - 1 \\ &= n^2(\beta+1) + n(\beta k - \beta) - \frac{k^2}{2} + \frac{3}{2}k - 1\end{aligned}$$

and

$$\begin{aligned}S_{n+1,M_k} &= \sum_{i=1}^{n} i + \beta n\left(k+n-1+\frac{n-k+1}{L}\right) \\ &= \frac{n(n+1)}{2} + \beta nk + \beta n^2 - \beta n + \frac{1}{L}(\beta n(n-k+1))\end{aligned}$$

We calculate that

$$\begin{aligned}\frac{S_{k,M_k}}{OPT} &\geq \frac{S_{k,M_k}}{S_{n+1,M_k}} \\ &\geq \frac{n^2(\beta+1)+n(\beta k-1)-\frac{k^2}{2}+\frac{3}{2}k-1}{\frac{n(n+1)}{2}+\beta nk+\beta n^2-\beta n+\frac{1}{L}(\beta n(n-k+1))} \\ &=: g(n,k,\beta,L)\end{aligned}$$

The approximation factor of any blind algorithm is bounded from below by lb with

$$lb := \max_{n,\beta,L} \min_{k} \max\{f(n,k,\beta), g(n,k,\beta,L)\}$$

By choosing $n := 1000$, $\beta := 1.96$, and L sufficiently large one can calculate that $\min_k \max\{f(n,k,\beta), g(n,k,\beta,L)\} \geq 1.1215$. The minimum value for k is obtained with $k = 439$. Thus, we have established our lower bound of 1.1215. □

7. Online Algorithms

In this subsection, we consider the increasing speed scheduling problem in an online setting. We assume the following online model:

- each job J_i has a release time r_i,
- the existence and all data of a job become known at its release time, and
- at time t the speed of the machine up to time t is known, the speed of the machine *after* time t is not known.

The Smith's rule algorithm in the online-setting works as follows: We always process the job which has the largest Smith's factor among all available jobs. Denote by $SR_{online}(I)$ the resulting schedule (and its objective function value) for an instance I. If $r_j = 0$ for all jobs then certainly $SR(I) = SR_{online}(I)$.

Theorem 30. In the online setting, the Smith's rule algorithm has a competitive factor of 2.

Let I be an instance of our problem. With the same reasoning as in Lemma 21 we can show that w.l.o.g. we can assume that all $\frac{w_j}{\ell_j} = 1$ for all jobs in I. However, note that since we now allow the preemption of jobs, in the worst possible Smith's rule schedule a job J_i might be preempted at any time by a job $J_{i'}$, even if $\frac{w_{i'}}{\ell_{i'}} = \frac{w_i}{\ell_i}$.

We choose ε such that it divides the demand of each job and each release time. Now we create a lower bound instance I' by replacing each job J_i by ℓ_i/ε jobs, each with demand ε and weight $\frac{w_i}{\ell_i}\varepsilon$.

Lemma 31. *It holds that $OPT(I') \leq OPT(I)$.*

Proof. We start with $OPT(I)$. We show that splitting the jobs into pieces of demand ε does not increase the value of the objective function. Denote by S the schedule obtained by taking $OPT(I)$ and splitting each jobs J_i into ℓ_i/ε equal jobs with demand ε. Let $J_i \in \mathcal{J}$ denote a job which is executed in the time interval $[s_i, t_i)$ in $OPT(I)$. Then J_i contributes $w_i \cdot t_i$ towards the sum of weighted completion times. Denote by $\mathcal{J}_i$ the jobs resulting from splitting J_i. For a job $J_{i'} \in \mathcal{J}_i$ denote by $t(i')$ its completion time in S. In S the jobs resulting from splitting J_i contribute

$$\begin{aligned}\sum_{J_{i'} \in \mathcal{J}_i} t(i') \cdot \frac{w_i}{\ell_i}\varepsilon &\leq \sum_{J_{i'} \in \mathcal{J}_i} t_i \cdot \frac{w_i}{\ell_i}\varepsilon \\ &= \frac{\ell_i}{\varepsilon} \cdot t_i \cdot \frac{w_i}{\ell_i}\varepsilon \\ &= t_i \cdot w_i\end{aligned}$$

towards the objective function. Applying this reasoning to all jobs shows that $S \leq OPT(I)$. This implies that $OPT(I') \leq OPT(I)$. □

Lemma 32. *The schedule $SR_{online}(I')$ is optimal.*

Proof. Due to the choice of ε no job is ever preempted. The claim follows from Lemma 19. □

We observe that the time until the last job finishes can be partitioned into time intervals in which the machine has no idle time. Denote by $\mathcal{L}$ the set of these intervals. We now show the competitive factor of 2 for the Smith's rule algorithm.

Proof of Theorem 30. We consider each interval $L = [x, y] \in \mathcal{L}$ separately. Let $\mathcal{J}_L \subseteq \mathcal{J}$ be the jobs which finish within L in I and let $\mathcal{J}'_L \subseteq \mathcal{J}'$ be the jobs which finish within L in I'. We denote by $cost(\mathcal{J}_L)$ and $cost(\mathcal{J}'_L)$ the value that the jobs in $\mathcal{J}_L$ and $\mathcal{J}'_L$ contribute towards the objective function in I and I', respectively. We want to show that $cost(\mathcal{J}_L) \leq 2 \cdot cost(\mathcal{J}'_L)$.

We define a function f as follows: Assume that a job starts its execution at time x and is never interrupted. We define $f(\ell)$ to be its completion time depending on its demand. Note that since the machine does never slow down we have that

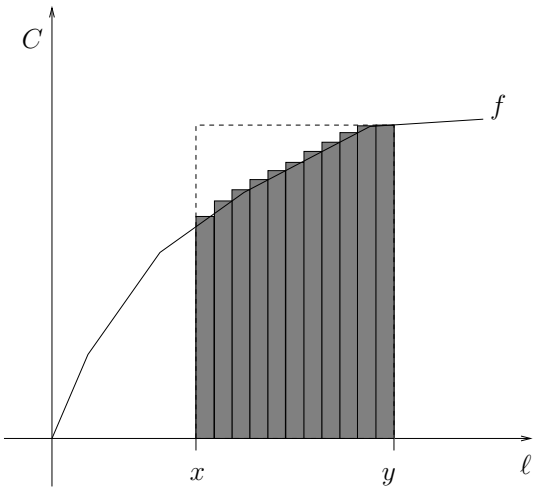

FIGURE 7.1. The function f together with the scheduling of the jobs in $\mathcal{J}_L$ and $\mathcal{J}'_L$. The narrow bars represent the value of our lower bound. The dotted line represents our upper bound on the value for the real objective function.

f is concave. We define $k := \frac{1}{\varepsilon}\sum_{j\in\mathcal{J}_L} \ell_j$, i.e., the schedule I' executes k jobs the interval L. We want to show that $cost\,(\mathcal{J}_L) \leq 2 \cdot cost\,(\mathcal{J}'_L)$. We calculate that

$$\begin{aligned} cost\,(\mathcal{J}'_L) &= \sum_{i=1}^{k} \varepsilon f\,(i\cdot\varepsilon) \\ &\geq \int_0^{k\varepsilon} f(z)\;dz \\ &\geq \frac{f\,(k\varepsilon)+f\,(0)}{2}\cdot k\varepsilon \end{aligned}$$

(see Figure 7.1 for a sketch) and thus

$$\begin{aligned} cost\,(\mathcal{J}_L) &\leq \sum_{j\in\mathcal{J}_L} f\,(k\varepsilon)\cdot w_j \\ &= f\,(k\varepsilon)\cdot\sum_{j\in\mathcal{J}_L}\ell_j \\ &= f\,(k\varepsilon)\cdot k\varepsilon \\ &\leq f\,(k\varepsilon)\cdot k\varepsilon + f\,(0)\cdot k\varepsilon \\ &= 2\cdot cost\,(\mathcal{J}_{L'}) \end{aligned}$$

Doing this reasoning for each interval $L \in \mathcal{L}$ proves the claim. □

Note that our analysis is tight since there are examples (even for the case that the speed of the machine does not change) where the online Smith's Rule algorithm does not perform better than factor 2 [13].

7.1. **Lower Bound for Online Algorithms.** The lower bound construction presented in Section 6 carries over to a bound for online algorithms. It actually also holds if all jobs are released at time $t = 0$ and only the information about the machine becomes available online.

Theorem 33. *No online algorithm for the increasing speed scheduling problem can achieve a better competitive ratio than* 1.1215*, even if all jobs have the same release time.*

Proof. Like in the proof of Theorem 29 we use the following set of jobs: We have n jobs with $w_0 = \beta \cdot n$, $\ell_0 = n$, $w_1 = w_2 = ... = w_n = 1 = \ell_1 = \ell_2 = ... = \ell_n$. The adversary runs the machine with unit speed until the job J_0 has completely been processed. Assume that the job J_0 was the kth job which was scheduled. Let L be a large constant to be defined later. We define $f(n, k, \beta)$ and $g(n, k, \beta, L)$ as in the proof of Theorem 29. If $f(n, k, \beta) \geq g(n, k, \beta, L)$ then the adversary continues to run the machine with unit speed after J_0 is finished. If $f(n, k, \beta) < g(n, k, \beta, L)$ then once J_0 is finished the adversary accelerates the machine to speed L. With the same reasoning as in the proof of Theorem 29 we obtain that the competitive ratio of the algorithm is at best 1.1215 for suitable choices of n, β, and L. □

7.2. **Unit Weight Case, Online.** In this section we prove that for the unit weight case the shortest remaining processing time algorithm (SRPT) is optimal (we will define the algorithm in the sequel). This extends the fact that SRPT is optimal for the problem $1|r_i, pmtn| \sum C_j$.

The SRPT-algorithm works as follows: at each point in time, we process the available job which has the shortest remaining demand. Ties are broken arbitrarily. For an instance I denote by $SRPT(I)$ the resulting schedule.

Theorem 34. *If all jobs in an instance I have the same weight then $SRPT(I)$ is optimal.*

Proof. Assume on the contrary that $SRPT(I)$ is not optimal. Let $OPT(I)$ be the optimal schedule which differs from $SRPT(I)$ for the first time as late as possible. Let time t be the first timestep where $OPT(I)$ and $SRPT(I)$ differ. Let J_{long} denote the job which is processed by $OPT(I)$ at time t and let J_{short} be the job which is processed by $SRPT(I)$ at time t. Then at time t the remaining demand of J_{short} is strictly shorter than the remaining demand of J_{long}. (If both remaining demands equal then by an exchange argument we can show that there must be an optimal schedule which also schedules J_{short} at time t.) Let t_{short} and t_{long} denote the finishing times of J_{short} and J_{long} in $OPT(I)$. Let I denote the union of the time intervals after t in which $OPT(I)$ processes either J_{short} or J_{long}. We define a new schedule $OPT'(I)$ as follows: outside I the schedules $OPT'(I)$ and $OPT(I)$ are identical. Within I the schedule $OPT'(I)$ processes J_{short} and J_{long} according to the SRPT-rule. Denote by t'_{short} and t'_{long} the finishing times of J_{short} and J_{long} in $OPT'(I)$, respectively. If $t_{short} < t_{long}$ then $t'_{short} < t_{short}$ and $t'_{long} = t_{long}$. If $t_{short} > t_{long}$ then $t'_{short} < t_{long}$ and $t'_{long} = t_{short}$. In both cases we derive that $t_{short} + t_{long} > t'_{short} + t'_{long}$. The finishing times of all other jobs are the same in both schedules. This contradicts that $OPT(I)$ is an optimal schedule. □

8. Acknowledgements

We are grateful to Ekki Köhler for the discussion at Schloss Dagstuhl in which the Flow scheduling problem was conceived.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pages 32–44. IEEE, 1999.

[2] N. Baumann and M. Skutella. Earliest arrival flows with multiple sources. *Mathematics of Operations Research*, 34:499–512, 2009.

[3] L. Fleischer. Faster algorithms for the quickest transshipment problem with zero transit times. In *Proceedings of the 9th Annual Symposium on Discrete Algorithms (SODA 1998)*, pages 147–156, 1998.

[4] D. Gale. Transient flows in networks. *Michigan Mathematical Journal*, 6:59–63, 1959.

[5] E. Gawrilow, E. Köhler, R. H. Möhring, and B. Stenzel. Dynamic routing of automated guided vehicles in real-time. In *Mathematics — Key Technology for the Future*, pages 165–178. Springer, 2008.

[6] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. In *Proceedings of the 30th Annual International Colloquium on Automata, Languages and Programming (ICALP 2003)*, pages 397–409, 2003.

[7] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 379:387–404, 2007.

[8] H. Hoogeveen, M. Skutella, and G. J. Woeginger. Preemptive scheduling with rejection. In *Prceedings of the 8th European Symposium on Algorithms (ESA 2000)*, volume 1879 of *Lecture Notes in Computer Science*, pages 268–277. Springer, 2000.

[9] B. Hoppe and É. Tardos. Polynomial time algorithms for some evacuation problems. In *Proceedings of the 5th Annual Symposium on Discrete Algorithms (SODA 1994)*, pages 433–441, 1994.

[10] B. Hoppe and É. Tardos. The quickest transshipment problem. *Math. Oper. Res.*, 25(1):36–62, 2000.

[11] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984.

[12] E. Minieka. Maximal, lexicographic, and dynamic network flows. *Operations Research*, 21:517–527, 1973.

[13] A. S. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, pages 121–133, 2002.

[14] W. E. Smith. Various optimizers for single-stage production. *Naval Research and Logistics Quarterly*, pages 59–66, 1956.

[15] W. L. Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operations Research*, 19:1602–1612, 1971.