Make-to-Order Integrated Scheduling and Distribution *

Yossi Azar† Amir Epstein‡ Łukasz Jeż§ Adi Vardi¶

Abstract

Production and distribution are fundamental operational functions in supply chains. The main challenge is to design algorithms that optimize operational performance by jointly scheduling production and delivery of customer orders. In this paper we study a model of scheduling customer orders on multiple identical machines and their distribution to customers afterwards. The goal is to minimize the total time from release to distribution plus total distribution cost to the customers. We design the first poly-logarithmic competitive algorithm for the problem, improving upon previous algorithms with linear competitive ratios. Our model generalizes two fundamental problems: scheduling of jobs on multiple identical machines (where the goal function is to minimize the total flow time) as well as the TCP Acknowledgment problem.

1 Introduction

Production and distribution are fundamental operational functions in supply chains. Scheduling of production and distribution operations is necessary in many practical situations, where firms aim to optimize the tradeoff between various costs, total revenue, and delivery lead time (time between release and delivery), which reduces inventory levels and increases responsiveness to customers. Moreover, an increasing number of companies adopt make-to-order business models in which products are custom-made and delivered to the customers directly from the factory within a relatively short lead time. Hence, there is little to no inventory of products. For example, consider recently developed digital manufacturing (e.g., 3D printing technology) that enables highly customized products at lower volume and lower cost. Therefore, production starts only after orders have been placed. In addition, large retail companies such as Amazon sell about half a billion different products. Hence, they cannot keep a large inventory for each product.

In the last decades, a tremendous amount of research has been conducted on integrated production and distribution models. The study of these models at the detailed scheduling level is fairly recent. These models attempt to optimize operational performance by jointly scheduling production and delivery of customer orders. The scheduling decisions take into account cost, revenue and customer service level for each order.

Most integrated production and distribution scheduling (IPODS) models studied in the literature consider the offline setting, where full information about future orders (jobs) is known in advance. However, this assumption is non-realistic in many settings such as in make-to-order business models.

Therefore, in this paper we study an online version of the problem. In our model, there are n customers that are to be served by the manufacturer's m identical machines. Each customer has jobs (orders) that he releases over time, where each job has a processing time and can be preempted. The manufacturer must process jobs and deliver finished jobs (products) to the customer. Once a job has been completed, it can be delivered to the customer, but not necessarily immediately. Each customer has a non-negative delivery cost that does not depend on the number of jobs delivered in the same delivery (batch). Furthermore, deliveries to different customers cannot be combined. The goal is to minimize the total cost, which is the sum of the total lead time and the total delivery cost. To this end, the manufacturer may postpone the delivery of completed jobs and aggregate them in batches. We note that the lead time can also be viewed as the holding cost incurred over time.

Interestingly, this model is a generalization of two fundamental problems in computer science that have received a lot of attention. When there are no delivery costs (i.e., all delivery costs equal zero), the problem

*Supported in part by the Israel Science Foundation and by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), NWO Vidi grant 639.022.211, and Polish NCN grant DEC-2013/09/B/ST6/01538.

†School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: azar@tau.ac.il

‡IBM Research-Haifa, Haifa, Israel. Email: amire@il.ibm.com

§Eindhoven University of Technology, Netherlands, and Institute of Computer Science, University of Wrocław, Poland Email: lje@cs.uni.wroc.pl

¶School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: adi.vardi@gmail.com.

reduces to the scheduling problem of minimizing the total flow time on parallel identical machines, see [5, 19]. On the other hand, when all jobs have zero processing time, the problem reduces to the TCP Acknowledgment problem for each customer separately, see [13, 14].

1.1 Our and Previous Results

We provide the first poly-logarithmic competitive algorithm for the integrated production and distribution scheduling problem. Specifically, our main result is an $O(\log m + \log \Delta + \log \rho)$-competitive algorithm[1], where Δ and ρ denote the ratio of the maximum to minimum delivery cost and job processing time, respectively (recall that m is the number of machines). We complement it with a lower bound that, together with those previously known, implies that our algorithm is essentially optimal.

Our results exponentially improve the previous best results even for a single machine. Specifically, for a single machine our algorithm is $O(\frac{\log \Delta}{\log \log \Delta} + \log \rho)$-competitive, whereas previous results were (at least) linear in either parameter. Specifically, Averbakh et al. [3] gave an online algorithm for the case of a single machine that achieved an upper bound depending on all customers' delivery costs[2] that is no better than $\Omega(n + \Delta)$. Averbakh et al. [2] provided an online algorithm that achieved an upper bound of $\rho + 3$ for the problem.

For the special case of a single machine, where all jobs of each customer belong to a single class, we show an $O(\frac{\log \Delta}{\log \log \Delta})$-competitive algorithm (this can be extended to a fixed number of classes).

In addition, we show a lower bound of $\Omega(\frac{\log \Delta}{\log \log \Delta})$ for the problem, which holds even in the single machine environment, thus refuting the conjecture of Averbakh et al. [2] that a constant-competitive algorithm exists. Moreover, we note that lower bounds of $\Omega(\log \rho)$ and $\Omega(\log m)$ follow from the problem of minimizing flow time on identical machines [19], which is a special case of our problem with no delivery costs. Together, the lower bounds of $\Omega(\log m)$, $\Omega(\log \rho)$, and $\Omega(\frac{\log \Delta}{\log \log \Delta})$ imply that our algorithm is almost optimal[3].

As noted, if all delivery costs are zero, our problem simplifies to flow time minimization. In such a case, our algorithm becomes the classic non-migratory algorithm for minimizing flow time. Slightly more generally, if the processing time of a job is larger than its delivery cost, then the job can be delivered immediately, losing a factor of at most 2 in the competitive ratio. On the other hand, when all jobs have zero processing time, the problem reduces to the TCP Acknowledgment problem [13, 14] for each customer separately.

1.2 Techniques

Any online algorithm for our model must have two components. Job scheduling (i.e., which jobs to process at each point of time) and delivery scheduling (i.e., when to deliver jobs to the customer). These components cannot be decoupled: Specifically, on the one hand, we prefer to process jobs with small processing time in order to reduce the flow time (time between the release of the job and the time it is completed). On the other hand, we prefer to process jobs from the same client (especially if the delivery cost of this client is relatively large) in order to deliver them together in a single batch. Hence, the main challenge is to balance between the flow time, lag time (time between a job being completed and delivered) and delivery cost.

For a single machine with a single customer, Averbakh et al. [3] proposed the following elegant 2-competitive algorithm, abbreviated as $SRPT_D$. The jobs are scheduled according to SRPT (shortest remaining processing time). Each time the total lag time of completed undelivered jobs equals the delivery cost, these jobs are delivered to the customer.

Using $SRPT_D$ for multiple customers (even for a single machine) seems to yield a linear competitive ratio, see [3]. Hence, we develop a different algorithm, which nevertheless uses $SRPT_D$ to create batches of jobs for each customer. For job scheduling though, our algorithm uses the non-migratory scheduling algorithm presented in [5], which is known to achieve the best possible competitive ratio for flow time minimization. This algorithm partitions the jobs into classes according to job size. It does not specify how to choose between jobs of the same class. As we aim to decrease the number of interleaved batches, our specific realization of the algorithm activates a single batch from each class on a machine, and processes only the jobs from active batches.

We note that when the processing schedule is given, the deliveries can be taken care of in a nearly optimal way. Specifically, one can easily observe that delivering all completed undelivered jobs of each customer once they accumulate a lag time equal to the delivery cost of that customer yields a 2-competitive delivery schedule

[1] We assume without lost of generality that $\Delta \geq 4$, otherwise our algorithm uses $\Delta = 4$.

[2] Specifically, its guarantee is $2 \sum_i D_i / \min_i D_i$, where D_i is the i-th customer's delivery cost. Hence, this guarantee is bounded by $\Omega(n + \Delta)$ and $O(n\Delta)$.

[3] Note that with multiple parameters, optimality is not uniquely defined since the lower bounds hold for a certain combination of parameters.

(TCP Acknowledgment [13, 14]). However, it is difficult to analyze the combined algorithm (scheduling and delivery), since the job schedule produced by our algorithm may be far from optimal. Therefore, we analyze a "virtual" delivery procedure described below.

The main challenge of the delivery scheduling is handling interleaved batches. Although in $SRPT_D$ simulation each batch accumulates a lag time which equals the customer delivery cost, the actual accumulated lag time of jobs in the batch as processed by our algorithm may be large due to interleaving with batches of other customers. Note that a batch is not completed if it is preempted by other batches. Our algorithm handles this problem as follows. While the number of undelivered jobs in a batch is large, each time that the number of active jobs (jobs released but not yet completed) is close to the number of completed undelivered jobs, all these jobs are delivered to the customer. Therefore, the lag time of the completed undelivered jobs can be charged to the flow time of those still active. Moreover, the number of deliveries per batch is poly-logarithmic in the number of machines and the ratios of maximum to minimum processing time of the jobs and customers delivery costs.

When the number of active jobs in a batch drops below a certain threshold, the delivery pattern changes. Namely, we partition the batch into sub-batches consisting of jobs of the same size class (i.e., of roughly the same size). Then, there is only a single delivery per sub-batch, once all of its jobs are completed. We charge the lag time of these jobs to the flow time and delivery cost of other batches using an exponential charging scheme. We believe that this charging scheme can be used in related problems. Specifically, we charge each completed undelivered job to the flow time and delivery cost of all jobs from lower classes (since a job can be preempted only by a job from a lower class). A job from a certain class is charged with an appropriate weight, exponentially decreasing with the difference between the classes of the completed undelivered job and the preempting jobs.

Our lower bound is based on the following idea. There are only two customers, one with unit delivery cost and unit processing time jobs, and one with large delivery cost and jobs with large processing time. The lower bound consists of up to a logarithmic number of phases. The number of released unit processing time jobs increase exponentially with the phase, while the number of remaining large jobs decreases exponentially with the phase. If there ever is a moment such that, due to prioritizing the large jobs, the online algorithm has significantly more jobs than an algorithm that prioritizes small jobs, than the ratio between the number of their jobs can translated into a lower bound on the competitive ratio, as the former can be maintained by issuing a unit processing time job at every time unit for a substantial amount of time. Naturally, this results in the same ratio between the flow times of the algorithms, and in such case the overall cost is dominated by the flow time. Thus, a competitive online algorithm is forced to prioritize small jobs. As these arrive over time in successive phases, such algorithm starts processing the large jobs in each phase only to switch to the small jobs in the successive one. As the delivery cost for large jobs is large, the net result is that the algorithm accumulates large cost (lead time + delivery cost) for these jobs, which again dominates the overall cost.

1.3 Related work

Research on integrated production-distribution models at the detailed scheduling level is fairly recent, with the majority of the work done in the last several years. However, as a survey by Zhi-Long Chen [10] points out, its roots can be found in two early articles by Potts [20] and Cheng et al. [11]. In fact, the latter studies (from the offline perspective) a problem quite similar to ours: the only difference being that there is a single machine and a single customer, and instead of flow time, there is an "earliness penalty" in the objective, which applies if a job is delivered before its due date. While the nature of such problems motivates the study of online algorithms, the vast majority of the work concerns offline algorithms. The articles by Averbakh et al. [3, 2] (cf. Section 1.1 for details) are among the few exceptions. All other work on online algorithms in this context focuses on minimizing the maximum delivery time [22, 15, 1].

Integrated production-distribution problems have received even more attention at the strategic and tactical planning levels, and references to several surveys on these can be found in Chen's survey [10]. Because production and distribution stages at a planning level are often linked by an intermediate inventory stage, almost all studies in this area involve inventory decisions in addition to production and distribution decisions. We note that inventory problems, e.g., the Joint Replenishment Problem have recently been studied from the perspective of approximation and online algorithms [8, 7]. In fact, the TCP Acknowledgment problem is equivalent to the most fundamental of these, the Lot Sizing Problem [23]. See a short survey by Chrobak [12] for an overview of these recent results.

Recall that if there are no delivery costs, our problem is equivalent to flow time minimization. On a

single machine, the shortest remaining processing time (SRPT) algorithm, which schedules at any time the job with shortest remaining processing time, maximizes the number of jobs completed at any point in time [21], and hence minimizes the total flow time. For multiple machines, this is no longer the case. Nevertheless, SRPT is still an optimal online algorithm for flow time minimization. Specifically, Leonardi and Raz [19] proved that it is $O(\log \rho)$-competitive, and gave a matching lower bound of $\Omega(\log \rho)$. Moreover, they proved a lower bound of the logarithm of the ratio between the number of jobs and the number of machines, from which a lower bound of $\Omega(\log m)$ follows. Note that SRPT is a migratory algorithm. However, a non-migratory algorithm developed by Awerbuch et al. [5] is also $O(\log \rho)$-competitive in terms of flow time, see also [4, 9]. For other results on flow time, see e.g. [6, 17, 18, 16].

2 Preliminaries

There are n customers and m identical machines of a manufacturer. Jobs arrive online over time, where each job J_{ij} of client i has a release time r_{ij} and a processing time p_{ij}. The remaining processing time or volume of a job $J = J_{ij}$ at time t is denoted $p_{ij}(t)$ or $p(J,t)$. The manufacturer must deliver completed jobs to the customer. The cost of delivering any number of jobs (at once) to customer i is D_i; deliveries to different customers cannot be combined. We denote the minimum and maximum job processing time by $p_{\min}$ and $p_{\max}$ respectively, and let $\rho = \frac{p_{\max}}{p_{\min}}$. This notation is extended to sets of jobs, such as batches, with the obvious meaning of $p_{\max}(B)$, $p_{\min}(B)$, and $\rho(B)$. Similarly, we let $D_{\min} = \min_i D_i$, $D_{\max} = \max_i D_i$ and $\Delta = \frac{D_{\max}}{D_{\min}}$.

An online algorithm is not aware of job J_{ij} until time r_{ij}. At time r_{ij}, the algorithm learns p_{ij}. We assume that preemptions are allowed, where a job can be interrupted and resumed from the last point of execution. A schedule S specifies for each job when and on which machine it is processed and when it is delivered. The completion time of job J_{ij} (specifically, when processing of the job is completed) in schedule S is denoted by $C_{ij}(S)$. The flow time is $F_{ij}(S) = C_{ij}(S) - r_{ij}$. The total flow time is $F(S) = \sum_{i=1}^{n} \sum_j F_{ij}(S)$. The delivery time is denoted by $d_{ij}(S)$. The lag time is $d_{ij}(S) - C_{ij}(S)$ and the lead time is $L_{ij}(S) = d_{ij}(S) - r_{ij}$. The total lead time is $L(S) = \sum_{i=1}^{n} \sum_j L_{ij}(S)$. Note that the lead time is equal to the sum of flow time and lag time. Let $b_i(S)$ be the number of deliveries for customer i in the schedule S. The total delivery cost is $DC(S) = \sum_i b_i(S) D_i$. The objective is to minimize the sum of the total lead time $L(S)$ and the total delivery cost $DC(S)$. We sometimes omit S when the context is clear. We assume that transportation time of a delivery is 0, since non-zero transportation times would increase the value of each solution by a constant (per job), and could thus only improve the competitive ratio.

We say that a job is *active* at time t if by that time it has been released but not yet completed by the online algorithm. We also say that a job is *completed undelivered* at time t if by that time it has been completed but not yet delivered to the customer by the online algorithm.

Let I^x be an instance with n customers and x identical machines. If x is larger than 1 then each machine has a speed of 1, otherwise the machine has a speed of m. We let I_i^x be the projection of I^x on customer i, i.e., an instance where customer i is the only customer, with the same delivery cost D_i and the same set of jobs as in I^x. We let $A(I^x)$ denote the total cost of algorithm A and let $OPT(I^x)$ denote the total cost of the optimal solution on instance I^x, respectively. If I^x is clear from context, we omit it. We denote by S^* some optimal schedule. We denote by $N^A(t)$ the number of active jobs in algorithm A at time t. We also denote by $D^A(t)$ the number of completed undelivered jobs in algorithm A at time t. It is well known that the total flow time of algorithm (or schedule) A equals $\int N^A(t)\,\mathrm{d}t$. Similarly, the total lag time of algorithm A equals $\int D^A(t)\,\mathrm{d}t$. Thus, the total lead time of algorithm A equals $\int \left(N^A(t) + D^A(t)\right)\,\mathrm{d}t$. Thus, to prove relative bounds on the total lead time objective, it suffices to relate the total number of active or completed undelivered jobs of the algorithm and the benchmark, at every time t.

3 Multiple Machines

We partition the input jobs into classes according to their processing times as follows. Class k consists of all jobs J_{ij} with $p_k = \alpha^{k-1} \le p_{ij} < \alpha^k$, where $\alpha > 1$ is a constant. Thus, the total number of classes is at most $C = \lfloor \log_\alpha \rho \rfloor + 1 \le \log \rho + 1$. (In our algorithm, we implicitly assume $\alpha \ge 2$ and consequently we upper bound $\log_\alpha \rho$ by $\log_2 \rho$). Note that in some cases we refer to the original class of the job and in others to its current class, i.e., the class as determined by to the initial processing time and the current remaining one.

Our algorithm consists of three components: batching, scheduling, and delivery procedures. The first component is the batching procedure. Recall from Section 1.2 that $SRPT_D$, the 2-competitive algorithm of Averbakh et al. [3] for the single client and single ma-

chine setting, schedules the jobs according to SRPT. Each time that the total lag time of all the completed undelivered jobs exceeds the delivery cost, the algorithm delivers all the jobs.

The batching procedure (Procedure 1) simulates algorithm $SRPT_D$ for each customer i on a single machine with a speed of m. When jobs are delivered in the simulated algorithm, the procedure creates a batch that contains all these jobs, i.e., the batch is created when the lag time of all the completed undelivered jobs reaches D_i and consists of all jobs that are then delivered in the simulation. Each batch is partitioned into sub-batches according to job classes: For a batch B and a class k, let the k sub-batch of B, denoted B_k, be the set of all jobs from B of class k. We refer to a sub-batch of class k as k-batch. Note that the batching procedure keeps a single open batch per client (more jobs might be added to this batch). We refer to all other batches as *closed*. We assume that for each job j in a batch of client i, $p_{ij} \leq D_i$, otherwise the algorithm delivers job j immediately after its completion.

The second component is the job scheduling procedure (Procedure 2). This procedure is a specific realization of the non-migratory scheduling algorithm presented in [5], for job scheduling on multiple identical machines to minimize the total flow time. Their algorithm divides the input jobs into classes as described above, according to their remaining processing time. It maintains a stack of jobs for each machine and a global pool of yet unassigned jobs. The job to machine assignment is final, i.e., the jobs do not migrate. The algorithm works as follows: Each machine processes the job at the top of its stack. When a new job j arrives, if there exists an idle machine or a machine that processes a job from a higher class, j is pushed to the stack of that machine. Otherwise, j is inserted to the global pool. When a job has completed execution on machine M, it is removed from the stack, and a new job to be processed is chosen as follows: Let k be the minimum class of the jobs in the global pool. If the stack of machine M is empty or the job at the top of this stack belongs to a higher class than k, an arbitrary job from class k is pushed onto M's stack. Awerbuch et al. [5] proved that this non-migratory scheduling algorithm is $(\log \rho)$-competitive.

The scheduling procedure is a specific realization of the above algorithm. While their algorithm does not specify how to choose among jobs of the same class from the global pool, our specific realization, called NM_ALG, selects for each machine jobs from the active sub-batch of the same class of this machine. In particular, each machine has at most one active sub-batch from each class, but a single sub-batch may be assigned to multiple machines. Note that once the scheduling algorithm selected a sub-batch for execution, the sub-batch can be preempted only by sub-batches from a lower class. A sub-batch that was activated by the scheduling procedure is called an active sub-batch.

The last component is the delivery procedure (Procedure 3). It is simple to describe and implement but rather hard to analyze. Hence, we analyze Procedure 4 described below. Note that when the processing schedule is given, using Procedure 3 yields a 2-competitive delivery schedule. The reason is that deliveries to different customers are independent (i.e., the only dependence between customers is in the now fixed processing schedule), and minimizing the total lag time and delivery cost of a single customer given his job completion times corresponds to an instance of the TCP Acknowledgment problem. Hence, by using Procedure 3 instead of Procedure 4 we may lose a factor of at most 2.

The delivery procedure consists of three phases. In the first phase, the procedure delivers all the completed jobs of B upon B's closing. The second phase is performed in iterations, where in each iteration the completed jobs of B are delivered when the number of completed undelivered jobs in B reaches the number of active jobs in B. In the last phase, the procedure performs a final delivery for each remaining sub-batch of B, once all of its jobs are completed.

Procedure 1 BATCHING FOR CUSTOMER i

Opening:

If there is no open batch of customer i, **then** open a new batch and its sub-batches for each class k.

Classification:

When a job J_{ij} of processing time in class k of customer i arrives, put it in the k-batch of customer i's open batch.

Closing:

Simulate $SRPT(I_i^1)$.

When the total lag time of the jobs in the open batch reaches D_i, close the batch and its sub-batches.

Our main result is the following theorem, proved in the next subsection.

THEOREM 3.1. *The Algorithm is $O(\log m + \log \Delta + \log \rho)$-competitive.*

3.1 Algorithm Analysis

In this section we present the analysis. We prove the key lemma and defer proofs (and, in one case, statement) of other lemmas to Appendix A. We start

Procedure 2 SCHEDULING

Simulate NM_ALG(I^m).

Assignment:

When NM_$ALG(I^m)$ pushes a job of class k from the global pool onto the stack of machine i:

If there are no jobs from the current active batch of class k on this machine (denoted by $B_k(i,t)$) in the global pool, **then** deactivate the active $B_k(i,t)$-batch and activate a non-empty k-batch, preferring closed ones.

Push an arbitrary job from the active $B_k(i,t)$-batch onto the stack of machine i.

Processing:

Each machine processes the job at the top of its stack, removing it upon completion.

Procedure 3 DELIVERY

For each customer i:

Once the lag time of all the completed undelivered jobs of customer i equals D_i, *deliver* them to the customer.

Procedure 4 DELIVERY FOR A CLOSED BATCH

Phase 1:

When the batch B of customer i becomes closed (in Procedure 1), *deliver* all the completed jobs of B.

Phase 2:

While the number of active jobs in B is at least $\sqrt{\frac{D_{\min}}{p_{\max}(B)\cdot m}}$:

If the number of completed undelivered jobs in B becomes greater or equal to the number of active jobs in B, **then** *deliver* all the completed undelivered jobs of B.

Phase 3:

Once the number of active jobs in B is less than $\sqrt{\frac{D_{\min}}{p_{\max}(B)\cdot m}}$:

Deliver all the completed undelivered jobs of B.

For each k, **if** all the jobs of B_k are completed, **then** *deliver* all undelivered jobs of B_k.

with establishing simple lower bounds on the optimum. The following Lemma is based on the same idea as in [3].

LEMMA 3.1. $OPT(I^m) \geq \sum_i b_i(SRPT_D(I_i^1))D_i$.

As NM_ALG is a specific realization of the non-migratory scheduling algorithm, with competitive ratio of $O(\log \rho)$ for flow time minimization [5], it follows that

$$F(NM\_ALG(I^m))/OPT(I^m) = O(\log \rho). \tag{3.1}$$

We are going to use Lemma 3.1 and (3.1) as lower bounds on OPT.

As our scheduling procedure (Procedure 2) uses NM_ALG, it follows from (3.1) that our algorithm's flow time is an $O(\log \rho)$-approximation to the optimum flow time. It remains to consider the total lag time and delivery cost. First we note that the number of deliveries of our algorithm for a single batch can be bounded as follows.

LEMMA 3.2. *The total number of deliveries for a single batch B by our algorithm is at most* $\log m + (\log \Delta + 3\log \rho(B) + 9)/2$.

Next, we bound the lag time of jobs during phases 1 and 2 of the delivery procedure.

LEMMA 3.3. *Consider a batch B of customer i. The total lag time accumulated by the jobs of B during Phases* 1 *and* 2 *of the delivery procedure is at most the sum of the flow time accumulated by the jobs of B during this period and D_i.*

Finally, we prove the key Lemma that considers the total lag time of jobs delivered in Phase 3 of the delivery procedure. We bound it as a function of the flow time achieved by the Scheduling Procedure and the delivery cost of each customer on a unified machine.

LEMMA 3.4. *The total lag time associated with all the jobs delivered in Phase 3 of the delivery procedure is at most*

$$\frac{9}{2}F(NM\_ALG(I^m)) + 25(1+\log \rho)\sum_i b_i(SRPT_D(I_i^1))D_i.$$

Proof. Consider a batch B of customer i, and let t_0 denote the time of the last delivery for B in Phase 2 of Procedure 4. Let t_k denote the unique delivery time of the k-batch B_k in Phase 3. From now on we fix k and focus on the k-batch B_k.

We partition the interval $[t_0, t_k]$ into subintervals of length $\frac{9}{8}L$, where $L = 8\sqrt{D_{\min}p_{k+1}}$ (the last interval may be shorter). Note that the total lag time of the

jobs in the k-batch B_k within any such interval is at most $\frac{9D_{\min}}{\sqrt{m}}$, since there are at most $\sqrt{\frac{D_{\min}}{p_{k+1}\cdot m}}$ completed undelivered jobs in the k-batch B_k at time t_0. Next we show how to charge the lag time of completed undelivered jobs of sub-batch B_k in each subinterval T to the flow time of jobs processed by NM_ALG and delivery cost of jobs delivered by $SRPT_D$ to other customers in this interval.

If the last job of B_k is completed within T and thus the whole B_k delivered, we charge its lag time to the delivery cost of batch B in $SRPT_D(I_i^1)$; note that this delivery cost is $D_i \geq D_{\min}$. Since the batch B consists of at most $\log\rho+1$ sub-batches, the total charge to B's delivery is at most $9D_{\min}(\log\rho+1)/\sqrt{m}$, i.e., at most $9(\log\rho+1)/\sqrt{m}$ times its cost.

Otherwise, some jobs of B_k are active at the end of T, which implies that T has the full length of $\frac{9}{8}L$. Let q be any machine such that B_k is the active k-batch on the machine at the beginning of T. Note that since some jobs of B_k are active at the end of T and some of them are completed undelivered at the beginning of T, at least one machine satisfy the desired property. Since there are at most $\sqrt{\frac{D_{\min}}{p_{k+1}\cdot m}}$ uncompleted jobs of B_k when Phase 3 of Procedure 4 starts (and hence also when T starts), the total time they are processed within T on machine q is at most $\sqrt{\frac{D_{\min}}{p_{k+1}\cdot m}}\cdot p_{k+1} = \sqrt{\frac{D_{\min}p_{k+1}}{m}}$. Thus, jobs of other batches are processed within T on machine q for at least a total time of $\frac{9}{8}L - \sqrt{\frac{D_{\min}p_{k+1}}{m}} \geq L$. Recall that our scheduling procedure uses $NM\_ALG(I^m)$ algorithm. Consider the set of jobs from batches other than B_k that are processed in T on machine q by (our non-migrative algorithm) $NM\_ALG(I^m)$, and partition it into two subsets: the set X of jobs that arrived before T and the set Y of jobs that arrived during T. We will charge the lag time of the jobs in B_k to the flow time of jobs from X in $NM\_ALG(I^m)$ or to the delivery cost of jobs from Y in $SRPT_D(I_{i'}^1)$ for appropriately chosen customers i'. We consider the following two cases.

Case 1: The total time $NM\_ALG(I^m)$ spent on processing the jobs of X within T on machine q is at least $L/2$. Let $T(X)\subseteq T$ be the subset of those time units in T in which $NM\_ALG(I^m)$ is processing jobs from X. Let τ be the median of $T(X)$, i.e., a time unit in $T(X)$ such that $|T_{<\tau}(X)| = |T_{\geq\tau}(X)| = |T(X)|/2 \geq L/4$, where $T_{<\tau}(X) = \{t\in T(X) \mid t<\tau\}$ and $T_{\geq\tau}(X)$ is defined analogously. Let $X'\subseteq X$ be the subset of jobs from X that $NM\_ALG(I^m)$ processes in $T_{\geq\tau}(X)$. Note that each job in X' accumulates a flow time of at least $|T(X)|/2 \geq L/4$ within $|T_{<\tau}(X)$ in $NM\_ALG(I^m)$.

Let $g(J,t)$ denote the class of job J according to its remaining processing time at time t in $NM\_ALG(I^m)$, i.e., $g(J,t) = \lceil \log p(J,t)\rceil$. In addition, let $X'_h = \{J \mid (J\in X')\wedge(g(J,\tau)=h)\}$, i.e., X'_h are the jobs of X' that belong to class h at time τ. Finally, let l_h be the total time that $NM\_ALG(I^m)$ is processing (the jobs of) X'_h within $T_{\geq\tau}(X)$. The jobs in X' are being charged to as follows. For each $J\in X'$ and each $t\in T_{<\tau}(X)$, the flow time accumulated by J between t and $t+\Delta t$ is charged $\alpha^{g(J,t)-k}\Delta t$ by the jobs of B_k.

Now we show that, within T, the charge from B_k to jobs in X' is least $4D_{\min}$. Clearly, within T, $NM\_ALG(I^m)$ is processing only jobs with remaining processing time at most $p_{k+1} = \alpha^k$ on machine q. Hence, $g(J,t)\leq k$, and the total charge to jobs in X' on account of B_k is at least

$$\begin{aligned}
&\sum_{J\in X'}\int_{T_{<\tau}(X)} \alpha^{g(J,t)-k}\,\mathrm{d}t \geq \sum_{J\in X'}\int_{T_{<\tau}(X)} \alpha^{g(J,\tau)-k}\,\mathrm{d}t\\
&= |T_{<\tau}(X)|\sum_{J\in X'}\alpha^{g(J,\tau)-k} \geq \frac{L}{4}\sum_{J\in X'}\alpha^{g(J,\tau)-k}\\
&= \frac{L}{4}\sum_{h|h\leq k}\sum_{J\in X'_h}\alpha^{h-k} = \frac{L}{4}\sum_{h|h\leq k}|X'_h|\alpha^{h-k}\\
&\geq \frac{L}{4}\sum_{h|h\leq k} l_h\alpha^{-k} = \frac{L}{4}\alpha^{-k}\sum_{h\leq k} l_h \geq \left(\frac{L}{4}\right)^2\alpha^{-k}\\
&= 4D_{\min}p_{k+1}\alpha^{-k} = 4D_{\min}\ .
\end{aligned}$$

The first inequality follows from the fact that the function g is non-increasing in t. The second inequality follows since $|T_{<\tau}(X)|\geq\frac{L}{4}$. The third inequality follows since $|X'_h|\geq l_h\alpha^{-h}$, as each $J\in X'_h$ satisfies $g(J,\tau)=h$, and hence $p(J,\tau)\leq\alpha^h$, and these jobs are processed for l_h units of time in $T_{\geq\tau}(X)$.

We note that a unit of job $J\in X'$ running at time t may be charged this way by active k-batches of all customers for all $k\geq g(J,t)$. Scaled by a factor of $\frac{9}{4}$, the total charge to such unit is thus at most

$$\frac{9}{4}\sum_{k\geq g(J,t)}\alpha^{g(J,t)-k} \leq \frac{9}{4}\sum_{j=0}^{\infty}\alpha^{-j} = \frac{9}{4}\frac{\alpha}{\alpha-1}\ .$$

We conclude that the total charge from all active k-batches to flow time of $NM\_ALG(I^m)$ is at most $\frac{9\alpha}{4(\alpha-1)\sqrt{m}}F(NM\_ALG(I^m))$.

Case 2: The total processing length of jobs in Y within T on machine q is at least $L/2$. Let l'_h denote the total length of jobs of class h at their release time (i.e., their "original" class) that are being processed in this interval. (Note that this is in contrast to l_h defined in the first case, where we used the current class.) Each of these jobs belongs to a batch that was created in $SRPT_D(I_{i'}^1)$ and is charged as follows. Each job of class

h is charged $\alpha^{h-k/2}\sqrt{\frac{D_{\min}}{m}}$ at its release time by jobs of class k from B_k. Note that Y is composed of jobs from class at most k, since algorithm $NM\_ALG(I^m)$ processed them rather than a job from class k. The total number of jobs from class $h \le k$ is at least $\frac{l'_h}{\alpha^h}$ and each of them is charged $\alpha^{h-k/2}\sqrt{\frac{D_{\min}}{m}}$ by jobs from B_k in T. Thus, the total charge from sub-batch B_k to jobs of Y is at least

$$\sum_{h\le k}\frac{l'_h}{\alpha^h}\alpha^{h-k/2}\sqrt{\frac{D_{\min}}{m}} = \alpha^{-k/2}\sqrt{\frac{D_{\min}}{m}}\sum_{h\le k} l'_h$$
$$= \alpha^{-k/2}\sqrt{\frac{D_{\min}}{m}}\sum_{h} l'_h \ge \alpha^{-k/2}\sqrt{\frac{D_{\min}}{m}}\frac{L}{2}$$
$$\ge \alpha^{-k/2}\sqrt{\frac{D_{\min}}{m}}4\sqrt{D_{\min}\alpha^k} \ge 4\frac{D_{\min}}{\sqrt{m}} \ .$$

Now we bound the total charge to each job in Y by all active sub-batches of all customers. We sum up all the charges from all active k-batches to a job of class h in Y at its release time. Since there is at most one active batch from each class at each point of time (recall that all the jobs in Y are processed on machine q and that there is no migration), a job in $SRPT_D(I^1_{i'})$ can be charged in this way only by one active batch from each class. Thus, in this case each job from class h in $SRPT_D(I^1_{i'})$ is totally charged at most

$$\sum_{k=h}^{C}\alpha^{h-k/2}\sqrt{\frac{D_{\min}}{m}} = \sum_{k=h}^{C}\alpha^{(h-k)/2}\sqrt{\frac{D_{\min}\cdot\alpha^h}{m}}$$
$$\le \sqrt{\frac{D_{\min}\cdot\alpha^h}{m}}(\sum_{j=0}^{\infty}\alpha^{-j/2}) = \frac{\sqrt{\alpha}}{\sqrt{\alpha}-1}\sqrt{\frac{D_{\min}\cdot\alpha^h}{m}}.$$

Moreover, since the number of jobs in any class h sub-batch in $SRPT_D(I^1_{i'})$ is at most $\sqrt{\frac{2m\cdot D_{i'}}{p_h}}$ by (A.1), the total charge to sub-batch B_h in $SRPT_D(I^1_{i'})$ is at most

$$\sqrt{\frac{2m\cdot D_{i'}}{p_h}}\cdot\frac{\sqrt{\alpha}}{\sqrt{\alpha}-1}\sqrt{\frac{D_{\min}\cdot\alpha^h}{m}}$$
$$\le \sqrt{\frac{2m\cdot D_{i'}}{\alpha^{h-1}}}\cdot\frac{\sqrt{\alpha}}{\sqrt{\alpha}-1}\sqrt{\frac{D_{i'}\cdot\alpha^h}{m}} \le \frac{\alpha}{\sqrt{\alpha}-1}\sqrt{2}D_{i'}.$$

Recall that the total charge from sub-batch B_k to jobs of Y is at least $4\frac{D_{\min}}{\sqrt{m}}$. Hence, to cover the total delay of jobs in batch B, which is at most $\frac{9D_{\min}}{\sqrt{m}}$ in this case, we multiply the charge of each batch in $SRPT_D(I^1_{i'})$ by $\frac{9}{4}$. Thus, each sub-batch in $SRPT_D(I^1_{i'})$ is charged at most $\frac{9}{4}\cdot\frac{\alpha}{\sqrt{\alpha}-1}\sqrt{2}D_{\min}$. Since there are at most $\log\rho+1$ sub-batches per batch, the total charge from all active k-batches to batches of $SRPT_D(I^1_{i'})$ in this case is at most $\frac{9}{4}(\log\rho+1)\frac{\alpha}{\sqrt{\alpha}-1}\sqrt{2}D_{\min}b_{i'}(SRPT_D(I^1_{i'}))D_{i'}$. This complete Case 2.

Finally, we sum-up all the charges in the above cases to bound the total charge and thus the total lag time associated with all the final sub-batches. This completes the proof.

4 Single Machine

In this section we consider the case of a single machine ($m = 1$). For this case we devise a different algorithm and prove the following Theorem.

THEOREM 4.1. *There exists an online algorithm with competitive ratio $O(\frac{\log\Delta}{\log\log\Delta}+\log\rho)$ for a single machine.*

Moreover, we consider a special case where all the jobs of each customer belong to a single class (jobs of different customers may belong to different classes). For this case we prove the following. (Note that the algorithm, proofs, and a technical lemma are deferred to Appendix B.)

THEOREM 4.2. *There exists an online algorithm with competitive ratio $O(\frac{\log\Delta}{\log\log\Delta})$ for the case of a single machine, where all the jobs of each customer belong to a single class.*

5 Lower Bound

In this section we present a lower bound for our problem, proving the following.

THEOREM 5.1. *Any online algorithm has a competitive ratio of $\Omega(\frac{\log\Delta}{\log\log\Delta})$.*

Proof. For simplicity, we prove the bound only for deterministic algorithms and only for a single machine. In the end of the proof, we describe how to generalize it for randomized algorithms and for multiple machines. Note that the lower bound holds even for two customers, one with unit delivery cost and the other with delivery cost Δ.

The high level idea is as follows. There are only two customers, one with unit delivery cost and unit processing time jobs, and one with large delivery cost and jobs with large processing time. The lower bound consists of up to a logarithmic number of phases. Large jobs are released only initially. The unit processing

time jobs are released over phases. The number of the released unit processing time jobs increase exponentially with the phase, while the number of remaining large jobs decreases exponentially with the phase. If there ever is a moment such that, due to prioritizing the large jobs, the online algorithm has significantly more jobs than an algorithm that prioritizes small jobs, than the ratio between the number of their jobs can translated into a lower bound on the competitive ratio, as the former can be maintained by issuing a unit processing time job at every time unit for a substantial amount of time. Naturally, this results in the same ratio between the flow times of the algorithms, and in such case the overall cost is dominated by the flow time. Thus, a competitive online algorithm is forced to prioritize small jobs. As these arrive over time in successive phases, such algorithm starts processing the large jobs in each phase only to switch to the small jobs in the successive one. As the delivery cost for large jobs is large, the net result is that the algorithm accumulates large cost (lead time + delivery cost) for these jobs, which again dominates the overall cost.

We begin by describing the sequence. There are two customers: Customer A with $D_1 = 1$ and unit jobs (customer with unit jobs and unit delivery cost); Customer B with $D_2 = \Delta$ and jobs of processing time p (customer with "large" jobs and "large" delivery cost). The value of p will be defined later.

Sequence Structure: There are up to k phases, where phase i consists of $\sqrt{\Delta \cdot p}\left(c^i + \frac{1}{c^i}\right)$ time units (the values $c \geq 1$ and k will be defined later). Let $t_i^0 = t_{i-1}^0 + \sqrt{\Delta \cdot p}\left(c^i + \frac{1}{c^i}\right)$ denote the start time of phase i for $i \geq 1$ (where $t_0^0 = 0$). At the beginning of the first phase (i.e., at time t_0^0) $\sqrt{\Delta/p}$ jobs of customer B are released. Let $t_i^1 = t_i^0 + \sqrt{\Delta \cdot p}\left(\frac{1}{c^i} - \frac{1}{c^{i+1}}\right)$. For each phase i, where $0 \leq i \leq k-1$, a job of customer A is released at each time unit starting from time t_i^1 until the end of the phase. We refer to $[t_i^0, t_i^1)$ as the first time interval and to $[t_i^1, t_{i+1}^0)$ as the second time interval of phase i. Note that the length of the first time interval is $\sqrt{\Delta \cdot p}\left(\frac{1}{c^i} - \frac{1}{c^{i+1}}\right)$ and the length of the second time interval is $\sqrt{\Delta \cdot p}\left(c^i + \frac{1}{c^{i+1}}\right)$. We denote the online algorithm by ALG and the optimal offline algorithm by OPT. The exact sequence is defined in Figures 1 and 2.

We require that $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^k} \geq 1$ (otherwise we can not charge the lag time of the jobs in Step 2b of Figure 2).

Now we analyze the competitive ratio of ALG. Consider the following possible two cases (according to the termination type):

1. **Termination Case** 1**:** We begin with the following well known observation. Assume that in some point of time t_0 ALG has R times as many jobs as OPT does. Then a competitive ratio of R can be achieved in the following way. We release unit job at every time unit, ensuring that ALG has at least R times as many jobs as OPT at any point of time. Doing so for sufficiently long time forces a ratio of at least R on the flow time (recall that the flow time can be computed as the integral over the number of jobs) and makes other costs negligible. This follows since other costs are fixed and can be covered by the flow time of active jobs starting from time t_0 for sufficiently long time.

Let i be the index of the phase in which the sequence terminated. Hence, it is sufficient to prove that at the end of phase i, ALG has at least $R = O(\frac{\log \Delta}{\log \log \Delta})$ times as many jobs as OPT. Since the sequence was terminated by Termination

Phase i

1. **If** $i = 0$, **then** release $\sqrt{\Delta/p}$ jobs of customer B, each of processing time p.
2. Wait from t_i^0 to t_i^1, i.e., $\sqrt{\Delta \cdot p}\left(\frac{1}{c^i} - \frac{1}{c^{i+1}}\right)$ time units, the first time interval.
3. At time t_i^1 release a unit job of customer A at each time unit until time t_{i+1}^0, i.e., $\sqrt{\Delta \cdot p}\left(c^i + \frac{1}{c^{i+1}}\right)$ time units, the second time interval.

Figure 1: Phase i.

1. $i \leftarrow 0$
2. **While** $i < k$
   (a) Add phase i
   (b) **If** there are at most $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^{i+1}}$ active jobs of customer B at the end of the phase , **then** release a unit job at every time unit for sufficiently long time and terminate the sequence (Termination case 1).
   (c) $i \leftarrow i + 1$
3. Terminate the sequence (Termination Case 2).

Figure 2: Sequence Structure.

Case 1, then ALG has at most $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^{i+1}}$ active jobs of customer B at the end of this phase, i.e., a total volume of at most $\frac{1}{2}\frac{\sqrt{\Delta\cdot p}}{c^{i+1}}$. Recall that the total volume of these jobs in the beginning of the sequence is $\sqrt{\Delta\cdot p}$. Moreover, the sum of the lengths of the first time interval in phases 0 to i is $\sum_{j=0}^{i} t_j^1 - t_j^0 = \sum_{j=0}^{i}\sqrt{\Delta\cdot p}\left(\frac{1}{c^j}-\frac{1}{c^{j+1}}\right) = \sqrt{\Delta\cdot p}\left(1-\frac{1}{c^{i+1}}\right)$. Hence, the volume of jobs of customer B executed by the algorithm in the second time interval in phases 0 to i is at least

$$\sqrt{\Delta\cdot p}-\sqrt{\Delta\cdot p}\left(1-\frac{1}{c^{i+1}}\right)-\frac{1}{2}\frac{\sqrt{\Delta\cdot p}}{c^{i+1}} = \frac{1}{2}\frac{\sqrt{\Delta\cdot p}}{c^{i+1}}.$$

Since during each time unit of the second time interval a unit job of the customer A arrives, we conclude that there are at least $\frac{1}{2}\frac{\sqrt{\Delta\cdot p}}{c^{i+1}}$ active unit jobs at the end of phase i.

Let algorithm OPT' be as follows. For each $0 \le j \le i$, OPT' executes jobs of customer B during the first time interval of phase j, and jobs of customer A during the second time interval of that phase. Hence, OPT' executes each unit job of customer A upon its arrival. Moreover, since $t_j^1 - t_j^0 = \sqrt{\Delta\cdot p}\left(\frac{1}{c^j}-\frac{1}{c^{j+1}}\right)$, at the end of phase j, the volume of active jobs of customer B is $\sqrt{\Delta\cdot p} - \sum_{k=0}^{j}\left(t_k^1 - t_k^0\right) = \frac{\sqrt{\Delta\cdot p}}{c^{j+1}}$. In particular, at the end of phase i, the number of active jobs is $\frac{\sqrt{\Delta/p}}{c^{i+1}}$.

Finally we compute the ratio R of the number of jobs between ALG and OPT$'$, which satisfies:

$$R \ge \left(\frac{1}{2}\frac{\sqrt{\Delta\cdot p}}{c^{i+1}}\right) / \left(\frac{\sqrt{\Delta/p}}{c^{i+1}}\right) = \frac{p}{2}.$$

Later we choose $p = \Omega(\frac{\log\Delta}{\log\log\Delta})$ which completes the proof of this case.

2. **Termination Case 2:** First we analyze the cost of ALG. Since none of the phases satisfy Termination Case 1, for $0 \le i \le k-1$, there are at least $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^{i+1}}$ active jobs of customer B at the end of phase i. At each phase i, either there was a delivery to customer B (with delivery cost of Δ), or all the jobs that were active in the beginning of the phase accumulate lead time during the entire phase. Recall that the length of phase i is $\sqrt{\Delta\cdot p}\left(c^i+\frac{1}{c^i}\right) \ge c^i\sqrt{\Delta\cdot p}$. Hence, during each phase ALG accumulates a cost of at least $\min\{\Delta, \left(c^i\sqrt{\Delta\cdot p}\right)\left(\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^i}\right)\} = \frac{\Delta}{2}$. Since there are exactly k phases, the total cost satisfies:

$$ALG(I^m) \ge \frac{k\Delta}{2}. \tag{5.2}$$

Let OPT$'$ be the following algorithm. First OPT$'$ executes all the jobs of customer B and then delivers them to the customer. Then OPT$'$ executes all the jobs of customer A, where each job is delivered upon its completion.

Note that the length of the first phase (phase 0) is $2\sqrt{\Delta\cdot p}$. Moreover, at the beginning of the first phase $\sqrt{\Delta/p}$ jobs of customer B are released, and during the phase $\sqrt{\Delta\cdot p}(1+1/c)$ unit jobs of customer A are released. Hence, at the end of the first phase OPT$'$ delivers all the jobs of customer B, and is left with $\sqrt{\Delta/p}\cdot p + \sqrt{\Delta\cdot p}(1+1/c) - 2\sqrt{\Delta\cdot p} = \frac{\sqrt{\Delta\cdot p}}{c}$ active unit jobs. From now on, at every phase i, for $1 \le i \le k-1$, OPT$'$ reduces the number of active unit jobs of customer A to $\frac{\sqrt{\Delta\cdot p}}{c^{i+1}}$ during the first time interval.

Now we can upper bound the cost of OPT$'$. For phase i, $1 \le i \le k-1$ (not including phase 0), the length of the first time interval is $\sqrt{\Delta\cdot p}\left(\frac{1}{c^i}-\frac{1}{c^{i+1}}\right)$ and there are $\frac{\sqrt{\Delta\cdot p}}{c^i}$ active jobs at the beginning of the time interval. Hence the lead time accumulates in this time interval is at most:

$$\begin{aligned}&\left(\frac{\sqrt{\Delta\cdot p}}{c^i}\right)\sqrt{\Delta\cdot p}\left(\frac{1}{c^i}-\frac{1}{c^{i+1}}\right)\\ &\qquad = \left(\frac{\Delta\cdot p}{c^i}\right)\left(\frac{1}{c^i}-\frac{1}{c^{i+1}}\right).\end{aligned} \tag{5.3}$$

The length of the second time interval of that phase is $\sqrt{\Delta\cdot p}\left(c^i+\frac{1}{c^{i+1}}\right) \le 2c^i\sqrt{\Delta\cdot p}$. There are $\frac{\sqrt{\Delta\cdot p}}{c^{i+1}}$ active jobs at the beginning of the time interval, and a unit job arrives at each time unit (and is immediately completed and delivered). Therefore, the lead time accumulated at the second time interval is at most:

$$\begin{aligned}&\frac{\sqrt{\Delta\cdot p}}{c^{i+1}}2c^i\sqrt{\Delta\cdot p} + 2c^i\sqrt{\Delta\cdot p}\\ &\qquad = \frac{2\Delta\cdot p}{c} + 2c^i\sqrt{\Delta\cdot p}.\end{aligned} \tag{5.4}$$

Since OPT$'$ delivers each unit job upon its completion, the total delivery cost during the phase is equal to the number of jobs completed during that phase, which is equal to the length of the phase (since all jobs are of unit processing time). Therefore, the total delivery cost during the phase is:

$$\sqrt{\Delta\cdot p}\left(c^i+\frac{1}{c^i}\right) \le 2c^i\sqrt{\Delta\cdot p}. \tag{5.5}$$

We sum Equations (5.3), (5.4) and (5.5) to obtain the total cost (total lead time and total delivery cost) accumulated in the phase:

$$\begin{aligned}
&\left(\frac{\Delta\cdot p}{c^i}\right)\left(\frac{1}{c^i}-\frac{1}{c^{i+1}}\right)\\
&+\frac{2\Delta\cdot p}{c}+2c^i\sqrt{\Delta\cdot p}+2c^i\sqrt{\Delta\cdot p}\\
&\leq \frac{\Delta\cdot p}{c^{2i}}+\frac{2\Delta\cdot p}{c}+4c^i\sqrt{\Delta\cdot p}\\
&\leq \frac{\Delta\cdot p}{c}+\frac{2\Delta\cdot p}{c}+\frac{4\Delta\cdot p}{c}\leq\frac{8\Delta\cdot p}{c}, \qquad (5.6)
\end{aligned}$$

where the last inequality results from the fact that $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^k}\geq 1$.

Now we analyze the cost of the first phase (phase 0). The cost of the jobs of customer A can be bounded in a similar way as for the other phases. In particular, (5.6) holds for it, i.e., this is at most $\frac{8\Delta\cdot p}{c}$. Hence, we only need to analyze the cost of the jobs of customer B. At the beginning of the sequence there are $\sqrt{\Delta/p}$ jobs of customer B and they are delivered after $\sqrt{\Delta\cdot p}$ time units. Therefore, The lead time of jobs of customer B is $\sqrt{\Delta/p}\sqrt{\Delta\cdot p}=\Delta$ and the total delivery cost is Δ.

Summing the total cost over all the phases we obtain that the total cost of OPT satisfies:

$$\begin{aligned}
OPT(I^m)&\leq k\frac{8\Delta\cdot p}{c}+2\Delta\\
&=2\Delta\left(1+\frac{4p\cdot k}{c}\right). \qquad (5.7)
\end{aligned}$$

Combing Equations (5.2) and (5.7) we conclude that the competitive ratio is at least:

$$\left(\frac{k\Delta}{2}\right)/\left(2\Delta\left(1+\frac{4p\cdot k}{c}\right)\right)=\left(\frac{k}{4}\right)/\left(1+\frac{4p\cdot k}{c}\right).$$

We Set $p=k$, $c=k^2$ and $c^k=\frac{1}{2}\sqrt{\Delta/p}$ (recall that we required that $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^k}\geq 1$), i.e., $k^{2k}=\frac{1}{2}\sqrt{\Delta/k}$. Therefore $k=\theta\left(\frac{\log\Delta}{\log\log\Delta}\right)$ and the competitive ratio becomes:

$$\left(\frac{k}{4}\right)/\left(\frac{k^2+4k^2}{k^2}\right)=\frac{k^3}{4k^2+16k^2}=\Theta\left(\frac{\log\Delta}{\log\log\Delta}\right)$$

This completes the proof of this case.

We conclude that the competitive ratio is $\Omega(\frac{\log\Delta}{\log\log\Delta})$.

A generalization for randomized algorithms can be done as follows. The condition of Figure 2 step (2b) will be changed to:

> **If** with probability of at least 1/2 there are at most $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^{i+1}}$ active jobs of customer B at the end of the phase.

Figure 3: Changes to sequence structure

Then, in the analysis of Termination case 1 we bound the cost of ALG only where the event that there are at most $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^{i+1}}$ active jobs of customer B at the end of the phase occurs. In the analysis of Termination Case 2 we bound the cost of ALG only in phases such that there are at least $\frac{1}{2}\frac{\sqrt{\Delta/p}}{c^{i+1}}$ active jobs of customer B at the end of the phase. In expectation half of the phases satisfies this condition. Hence, in both cases we lose a factor of 2 in expectation.

A generalization for m machines can be done by duplicating each job in the sequence m times. Since the analysis is based on volume consideration, a similar analysis to the one used for a single machine can be applied.

6 Acknowledgements

We would like to thank David Breitgand for helpful and stimulating discussions about the problem.

References

[1] M. v. d. Akker, H. Hoogeveen, and N. Vakhania. Restarts can help in the on-line minimization of the maximum delivery time on a single machine. *Journal of Scheduling*, 3(6):333–341, 2000.

[2] I. Averbakh and M. Baysan. Approximation algorithm for the on-line multi-customer two-level supply chain scheduling problem. *Oper. Res. Lett.*, 41(6):710–714, 2013.

[3] I. Averbakh and Z. Xue. On-line supply chain scheduling problems with preemption. *European Journal of Operational Research*, 181(1):500–504, 2007.

[4] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 11–18. ACM, 2003.

[5] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. *SIAM Journal on Computing*, 31(5):1370–1382, 2002.

[6] L. Becchetti and S. Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 94–103. ACM, 2001.

[7] M. Bieńkowski, J. Byrka, M. Chrobak, Ł. Jeż, D. Nogneng, and J. Sgall. Better approximation bounds for the joint replenishment problem. In *Proc. of the 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 42–54, 2014.

[8] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Operations Research*, 61(4):1014–1029, 2013.

[9] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with ε resource augmentation. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 363–372. ACM, 2004.

[10] Z. Chen. Integrated production and outbound distribution scheduling: Review and extensions. *Operations Research*, 58(1):130–148, 2010.

[11] T. Cheng and H. Kahlbacher. Scheduling with delivery and earliness penalties. *Asia-Pacific Journal of Operational Research*, 10(2):145–152, 1993.

[12] M. Chrobak. Online aggregation problems. *SIGACT News*, 45(1):91–102, 2014.

[13] D. R. Dooly, S. A. Goldman, and S. D. Scott. TCP dynamic acknowledgment delay: Theory and practice (extended abstract). In *Proc. of the 30th Annual ACM Symp. on the Theory of Computing (STOC)*, pages 389–398, 1998.

[14] D. R. Dooly, S. A. Goldman, and S. D. Scott. On-line analysis of the TCP acknowledgment delay problem. *J. ACM*, 48(2):243–273, 2001.

[15] H. Hoogeveen and A. P. A. Vestjens. A best possible deterministic on-line algorithm for minimizing maximum delivery time on a single machine. *SIAM J. Discrete Math.*, 13(1):56–63, 2000.

[16] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000.

[17] B. Kalyanasundaram and K. R. Pruhs. Minimizing flow time nonclairvoyantly. *Journal of the ACM (JACM)*, 50(4):551–567, 2003.

[18] S. Leonardi. A simpler proof of preemptive total flow time approximation on parallel machines. In *Efficient Approximation and Online Algorithms*, pages 203–212. Springer, 2006.

[19] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007.

[20] C. N. Potts. Technical noteanalysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28(6):1436–1441, 1980.

[21] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In *Handbook of Scheduling - Algorithms, Models, and Performance Analysis.* 2004.

[22] S. S. Seiden. Randomized online scheduling with delivery times. *J. Comb. Optim.*, 3(4):399–416, 1999.

[23] H. Wagner and T. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5:89–96, 1958.

A Omitted proofs of Section 3.1

Proof of Lemma 3.1: We begin by noting that

$$OPT(I^m) \geq OPT(I^1) \geq \sum_i OPT(I_i^1) \ .$$

The first inequality holds since m machines with speed 1 can be simulated by a single machine with speed m (by running the m jobs "in parallel") in a way that does not increase completion times, allowing for same delivery times. The second inequality follows as deliveries of different customers cannot be combined, and the schedule of the original instance on a single machine can be simulated by the instance where each client runs on its own machine (of speed m).

To complete the proof, we show that the following holds for every customer i:

$$OPT(I_i^1) \geq F(SRPT(I_i^1)) + b_i(SRPT_D(I_i^1))D_i \ .$$

To this end, we use the fact that there exists an optimal solution for a single machine and a single costumer such that the jobs are scheduled according to SRPT algorithm [3]. This follows since SRPT maximizes the number of completed jobs at each point of time [21]. Hence, it can replace any other schedule without hindering the performance.

Consider $OPT(I_i^1)$ that uses SRPT for scheduling jobs. Its total flowtime is trivially the same as that of $SRPT_D$. Now consider any time interval between two consecutive deliveries of $SRPT_D(I_i^1)$. Then, either OPT performs a delivery within this interval, paying D_i, or its completed undelivered jobs at the end of the interval, being a superset of those of $SRPT_D$, have accumulated a total lag of at least D_i. This complete the proof.

Proof of Lemma 3.2: We begin by bounding the number of jobs in a batch of customer i (recall that the simulated machine of the batching procedure has a speed of m). Consider a batch B of customer i which consists of r_B jobs. We number these jobs $1, 2, \ldots, r_B$ in increasing order of completion time. Then in the batching procedure (Procedure 1), each job j accumulates a lag time of at least $\frac{1}{m}\sum_{j'=j+1}^{r_B} p_{j'} \geq \frac{1}{m}(r_B - j')p_{\min}(B)$, since it is already completed when the successive jobs are being processed. Hence, all these jobs accumulate a total lag time of at least $\sum_{j'=1}^{r_B} \frac{1}{m}(r_B - j')p_{\min}(B) \geq \frac{1}{m}p_{\min}(B) \cdot r_B(r_B-1)/2$. On the other hand, the batch B was closed when its total lag reached D_i, so it follows that

$$r_B \leq \sqrt{\frac{2D_i \cdot m}{p_{\min}(B)}} + 1 \leq 2\sqrt{\frac{2D_i \cdot m}{p_{\min}(B)}}. \tag{A.1}$$

The last inequality holds since $p_{\min}(B) \leq D_i$; recall that any job of customer i with processing time above D_i is delivered immediately after completion.

Next, we bound the number of deliveries for the batch during the delivery procedure. Note that after each delivery, there are no completed undelivered jobs in the batch. In particular, it follows from the previous paragraph that immediately after the first delivery (Phase 1), there are at most $r_B \leq 2\sqrt{\frac{2D_i \cdot m}{p_{\min}(B)}}$ active jobs in the batch. From now on, each successive delivery during Phase 2 reduces the number of active jobs by at least a factor of 2 until there are fewer than $\sqrt{\frac{D_{\min}}{p_{\max}(B)\cdot m}}$ active jobs. Hence, the number of such deliveries is at most

$$\begin{aligned}
&\left\lceil \log\left(r_B / \sqrt{\frac{D_{\min}}{p_{\max}(B)\cdot m}} \right) \right\rceil \\
&\leq \left\lceil \log\left(2 \cdot \sqrt{2 \cdot \frac{D_i}{D_{\min}} \cdot \frac{p_{\max}(B)}{p_{\min}(B)} \cdot m^2} \right) \right\rceil \\
&\leq \log m + (\log\Delta + \log\rho(B) + 5)/2 \ . \qquad \text{(A.2)}
\end{aligned}$$

Therefore, including the first delivery and up to $\log\rho(B) + 1$ final deliveries for the k-batches (the first delivery of Phase 3 has already been accounted for), the total number of deliveries for a single batch is at most $\log m + (\log\Delta + 3\log\rho(B) + 9)/2$. This complete the proof.

Next, we establish a relation between the lag time of a specific family of algorithms on m machines with speed of 1 and $SRPT_D$ on a single machine with speed of m.

LEMMA A.1. *Let A be any algorithm for the instance I^m such that the delivery times of all jobs of customer i are the same as in $SRPT_D(I_i^1)$. Then $\sum_j L_{ij}(A(I^m)) \leq \sum_j L_{ij}(SRPT_D(I_i^1))$.*

Proof. Since the delivery times of customer i are identical in both algorithms, the total lead time (total flow time and lag time) of customer i is equal in both algorithms. Hence, it is sufficient to prove that $\sum_j F_{ij}(A(I^m)) \geq \sum_j F_{ij}(SRPT_D(I_i^1))$. Recall that any schedule on m machines with speed of 1 can be simulated on a single machines with speed of m (such that the flow time on the single machine will be at most the flow time on the multiple machines). Since $SRPT$ achieves the optimal flow time on a single machine, its flow time on a single machine with speed m is at most the flow time of any schedule on m machines with speed 1. This completes the proof.

Proof of Lemma 3.3: Let $t_1, t_2, \ldots, t_b$ be the delivery times of batch B during phases 1 and 2 of the delivery procedure (Procedure 4). First we show that for every $1 < h \leq b$, the total lag time accumulated by the jobs of B in $[t_{h-1}, t_h]$ is no larger than the total flow time accrued by the jobs of B in $[t_{h-1}, t_h]$. The claim follows immediately from the description of Procedure 4: in each interval $[t_{h-1}, t_h]$, at any time t in the interval, the number of completed undelivered jobs from B is no larger than the number of active jobs from B.

Next, we show that the total lag time accumulated by the jobs of B before t_1 is no larger than D_i. According to Lemma A.1, the total lag time accumulated by the jobs of B by time t_1 in our algorithm is no larger than the total lag time accumulated by the jobs of B by time t_1 in $SRPT(I_i^1)$, which is exactly D_i by the third point of Procedure 1. This complete the proof.

B Omitted parts of Section 4

The algorithm is very similar to the one presented in the multiple machines section: we change the job scheduling procedure (Procedure 2) and make a small change to the delivery procedure (Procedure 4). The analysis of our algorithm resembles the analysis in the previous section. Hence, we focus mainly on the changes between the analysis. Since in this setting I_i^m coincides with I, we use the latter notation for convenience.

The changes to the algorithms are as follows. During phase 2 of the delivery procedure, instead of delivering the jobs each time that ratio between the completed undelivered jobs and the active jobs of a batch is at least 1, we deliver them each time that this ratio is at least $\lceil \frac{\log \Delta}{\log \log \Delta} \rceil$.

Instead of NM_ALG, the job scheduling procedure (Procedure 5) is a realization of algorithm $SRPT_\alpha$, which is based on SRPT but is oblivious to the different job processing times within a class (recall that the class is determined according to the initial processing time of the job). Algorithm $SRPT_\alpha$ simulates algorithm SRPT on the input. When SRPT schedules a job from class k, algorithm $SRPT_\alpha$ schedules a job from class k as follows. If there exists a job of class k with remaining processing time smaller than the minimum processing time of class k, the algorithm executes this job. Otherwise, the algorithm can process any active job from class k. We remark that the most natural variant of $SRPT_\alpha$ is one that is non-preemptive within each class. The general $SRPT_\alpha$ algorithm can be realized in many ways though. We note that all realizations of $SRPT_\alpha$ are $\lceil \alpha \rceil$-competitive for the total flow time objective function.

LEMMA B.1. *Algorithm $SRPT_\alpha$ is a $\lceil \alpha \rceil$-competitive algorithm for $1|r_i, pmtn| \sum F_i$.*

Proof. As noted in Section 2, it suffices to prove that $N^{SRPT_\alpha}(t) \leq \lceil \alpha \rceil N^{SRPT}(t)$ holds at every time t. In fact, we prove a slightly stronger claim, namely, that $N_k^{SRPT_\alpha}(t) \leq \alpha N_k^{SRPT}(t)$ holds for every time t and class k, where $N_k^A(t)$ denotes the number of active jobs from class k in algorithm A at time t.

To prove this, note that at each time t, $SRPT$ and $SRPT_\alpha$ process jobs from the same class. So fix a class k and let $W(t)$ denote the total remaining processing time of jobs of this class at time t in either algorithm. As each job in class k has processing time at most α^k, clearly, $N^{SRPT}(t) \geq \lceil W(t)\alpha^{-k} \rceil$. Moreover, as $SRPT_\alpha$ has at most one active job of class k with remaining processing time smaller than α^{k-1}, $N^{SRPT_\alpha}(t) \leq \lceil W(t)\alpha^{1-k} \rceil \leq \lceil \alpha \rceil \cdot \lceil W(t)\alpha^{-k} \rceil \leq \lceil \alpha \rceil \cdot N^{SRPT}(t)$.

Proof of Theorem 4.1: Recall that during phase 2 of the delivery procedure, the procedure delivers the jobs each time that the number of completed undelivered jobs in a a batch becomes greater or equal to the number of active jobs times $\lceil \frac{\log \Delta}{\log \log \Delta} \rceil$. Moreover, $m = 1$. Hence, the bound of Lemma 3.2 becomes $O(\frac{\log \Delta}{\log \log \Delta} + \log \rho(B))$. Moreover, the statement of Lemma 3.3 becomes: "Consider a batch B of customer i. The total lag time accumulated by the jobs of B during Phases 1 and 2 of the delivery procedure is at most the sum of the flow time accumulated by the jobs of B during this period times $\lceil \frac{\log \Delta}{\log \log \Delta} \rceil$ and D_i".

Lemma 3.4 analyzes a scheduling procedure that uses NM_ALG while our scheduling procedure uses $SRPT_\alpha$. Nevertheless, the proof is based on the following 3 properties: (i) Non-migratory, (ii) there is at most one active sub-batch per class on each machine, (iii) The processed job belongs to the lowest class among all active jobs (wrt current processing time). Since our scheduling procedure satisfies all 3 properties, we may use a similar analysis. Combining these bounds with Lemma B.1 and Lemma 3.1 proves Theorem 4.1.

Proof of Theorem 4.2: Recall that all the jobs of each customer belong to a single class. Hence, the term $\log \rho(B)$ in Lemma 3.2 can be replaced by $\log \alpha$: in this special case, we have $\rho(B) \leq \alpha$, and moreover, as each batch B consists only of a single sub-batch, there

Procedure 5 SINGLE MACHINE SCHEDULING

Simulate $SRPT(I)$. Let $k(t)$ be the class of the job (wrt its initial processing time) that $SRPT(I)$ processes at time t.

If there is no active k-batch, **then** activate a non-empty k-batch, preferring closed over open ones; note that a non-empty k-batch exists.

Process the job with the earliest arrival time from the active $B_k(t)$-batch.

If all jobs in the active $B_k(t)$-batch completed execution, **then** deactivate it.

Procedure 6 SINGLE MACHINE DELIVERY FOR A CLOSED BATCH

Phase 1:

When the batch B of customer i becomes closed (in Procedure 1), *deliver* all the completed jobs of B.

Phase 2:

While the number of active jobs in B is at least $\sqrt{\frac{D_{\min}}{p_{\max}(B)}}$:

If the number of completed undelivered jobs in B becomes greater or equal to the number of active jobs in B times $\lceil \frac{\log \Delta}{\log \log \Delta} \rceil$, **then** *deliver* all the completed undelivered jobs of B.

Phase 3:

Once the number of active jobs in B is less than $\sqrt{\frac{D_{\min}}{p_{\max}(B)}}$:

Deliver all the completed undelivered jobs of B.

For each k, **if** all the jobs of B_k are completed, **then** *deliver* all undelivered jobs of B_k.

is only one delivery in Phase 3 of Procedure 6. Thus, the upper bound of the Lemma becomes $O(\frac{\log \Delta}{\log \log \Delta})$.

Recall that we can use Lemma 3.4 although our scheduling procedure uses $SRPT_\alpha$. Since all the jobs of each customer belong to a single class, the bound becomes:

$$25 \sum_i b_i(SRPT_D(I_i))D_i + \frac{9}{2}F(SRPT_\alpha(I)) \ .$$

In addition, we use the statement of Lemma 3.3 for a single machine (see proof of Theorem 4.1). Combining these bounds with Lemma B.1 and Lemma 3.1 proves Theorem 4.2.