Operations Research Letters 41 (2013) 710–714

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

Approximation algorithm for the on-line multi-customer two-level supply chain scheduling problem

Igor Averbakh[*], Mehmet Baysan

Department of Management, University of Toronto Scarborough, 1265 Military Trail, Toronto, Ontario M1C 1A4, Canada

ARTICLE INFO

Article history:
Received 2 May 2013
Received in revised form
28 June 2013
Accepted 3 October 2013
Available online 11 October 2013

Keywords:
Supply chain scheduling
On-line algorithm
Competitive analysis
Integrated production–distribution problems

ABSTRACT

A manufacturer has to process jobs released on-line and deliver them to customers. Preemption is allowed. Jobs are grouped into batches for delivery. The sum of the total flow time and the total delivery cost is minimized. Deliveries to different customers cannot be combined. We present an on-line algorithm with the competitive ratio bounded by $3+\alpha$, where α is the ratio of the largest processing time to the smallest processing time.

1. Introduction and problem statement

In *supply chain scheduling problems* it is required to coordinate scheduling, batching, and delivery decisions, in order to minimize the overall scheduling and delivery cost. Supply chain scheduling problems were introduced by Hall and Potts [6] in the context of three-level supply chains that involve customers, manufacturers, and suppliers. Since then, various versions of supply chain scheduling problems have been studied (see the literature reviews in [1,2,5] and the survey [4]) in the off-line environment where full information about future jobs (orders) is available in advance (future is known).

In [1,2], two-level supply chain scheduling problems that involve a manufacturer and customers (also known as *integrated production–distribution problems* [1,4]) were studied in the on-line environment, where future is unknown. The basic setting considered in [1,2] and in this paper is as follows. Suppose that there are several customers $1,\dots,m$ and one manufacturer. Customer k releases jobs $J_{k1},\dots,J_{kp_k}$ on-line at times $r_{k1},\dots,r_{kp_k}$, $0 \le r_{k1} \le \cdots \le r_{kp_k}$, respectively. The jobs are released by means of phone or electronic orders so that the manufacturer can start processing a job as soon as it is released. At any instant, there is no information about the number, release and processing times of future jobs; the processing time of a job becomes known when the job is released. Processing times are positive. The manufacturer must process the jobs and deliver them back to the customers. The manufacturer can process no more than one job at a time; therefore, in the remainder of the paper the manufacturer will be called a *server*. Processed jobs will be called *completed* or *finished*. Preemption is allowed and does not require restarts; that is, a preempted job can be resumed from where it was stopped and does not have to be started from the scratch. To minimize the total cost, the jobs may not be delivered as soon as they are finished, and may wait until some other jobs are also finished, so that they can be delivered in batches. Deliveries to different customers cannot be combined (the case of "direct deliveries"). Let $D_i > 0$ be the cost of a delivery to customer i; it does not depend on the number of delivered jobs.

The *flow time* of a job is the time between its release and its delivery to the customer. We consider the problem of minimizing the sum of the total flow time of all jobs and the total delivery cost; this problem will be referred to as *Problem M*. Transportation time of a delivery is assumed to be 0 (nonzero transportation times would increase the total flow time of each solution by the same constant and would not change validity of our results).

An on-line algorithm A for Problem M is called ρ-*competitive*, for some $\rho \ge 1$, if for any problem instance the objective value of the solution obtained by algorithm A is bounded by ρ times the optimal off-line objective value for the instance plus a constant. (Since the scheduling problems that we consider are scalable, the additive constant can be deleted from the definition.) The *competitive ratio* of algorithm A is the infimum of the set of all values ρ such that algorithm A is ρ-competitive.

In [2], an on-line algorithm with competitive ratio 2 was presented for the single-customer version of Problem M ($m = 1$)

[*] Corresponding author.
E-mail addresses: averbakh@utsc.utoronto.ca (I. Averbakh), m.baysan@utoronto.ca (M. Baysan).

0167-6377/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.orl.2013.10.002

which will be referred to as *Problem S*, and it was shown that no on-line algorithm for the problem can have a better competitive ratio. For Problem M, an on-line algorithm was presented in [2] with an upper bound 2γ on the competitive ratio, where $\gamma = (\sum_{i=1}^{m} D_i)/(\min\{D_i, i \in [1 : m]\})$. In [1], the case of capacitated deliveries where each delivery can take no more than C jobs for some integer $C \geq 1$ was considered, and on-line algorithms were presented for capacitated Problems S and M and their modifications. The competitive ratio of the algorithm for the capacitated Problem S is 2; an upper bound $2\gamma'$ was obtained for the competitive ratio of the algorithm for the capacitated Problem M, where $\gamma' = \min\{C, 1 + (1 - \frac{1}{C})\frac{\sum_{i=1}^{m} D_i}{\min\{D_i, i \in [1:m]\}}\}$. Note that if deliveries to different customers have the same cost, then $2\gamma = 2m$ and $2\gamma' = 2 \cdot \min\{C, 1 + (1 - \frac{1}{C})m\}$; thus, the obtained performance guarantees for the algorithms from [1,2] for Problem M are quickly getting bad as the number of customers increases if the delivery capacity is not small.

The main contribution of this paper is an on-line algorithm for Problem M (with capacitated or uncapacitated deliveries) which is $(3 + \alpha)$-competitive, where α is the ratio of the processing time of the longest job to the processing time of the shortest job. The bound does not depend on m and provides reasonable performance guarantees for problems with limited variability in processing times.

We define the following terminology.

- A *full schedule* is a schedule that specifies both when jobs are processed and the times and contents of deliveries.
- A *job schedule* is a schedule that specifies only when jobs are processed but ignores deliveries.
- The *delay time* of a job is the time between the completion of the job and its delivery.
- The *currently active delay* (CAD) of a job at an instant between the completion of the job and its delivery is the time elapsed since the job's completion.
- The *finishing cost* of a job is the time between its release and completion.
- An *available* job is a job that has already been released but is not yet finished.
- A *normal* schedule is a schedule where the server cannot be idle if there are available jobs.

The flow time of a job is clearly the sum of its delay time and finishing cost. For any full schedule S, let $F(S)$ (respectively, $Q(S)$) denote the total finishing cost (respectively, the total delay time) of all jobs in S, and let $Z(S)$ denote the objective value for S (the sum of the total flow time and the total delivery cost).

The *shortest remaining processing time* (SRPT) rule prescribes to process at each instant t the job with the smallest remaining processing time among all already released unfinished jobs, with some unambiguous rule (e.g., lexicographic) for breaking ties. The SRPT rule can be implemented on-line. It is well known (see, e.g., [3]) that for the classical preemptive problem of minimizing the total flow time (which is equivalent to the special case of Problem M with $D_i = 0, i = 1, \ldots, m$), the SRPT rule provides an optimal solution.

Unless specified explicitly, deliveries are assumed to be uncapacitated; the capacitated case will be discussed separately in Section 5.

2. An algorithm and its performance

A *pseudo-schedule* for a customer i, denoted by PS(i), is an imaginary schedule of jobs of customer i and the corresponding deliveries, obtained by Algorithm PS(i) below for the imaginary situation where only the jobs of customer i are present.

Algorithm PS(i). Schedule the jobs of customer i according to the SRPT rule. Delivery to customer i is made as soon as the total CAD of finished undelivered jobs of customer i is equal to D_i; each delivery contains all finished undelivered jobs of customer i.

Algorithm PS(i) can be implemented on-line.

The delivery instants in PS(i) will be called *trigger instants* for customer i. For a trigger instant $\tau > 0$ for customer i, $J(\tau, i)$ will denote the set of jobs of customer i delivered at τ in PS(i). Clearly, a job can be preempted in PS(i) only at the release time of another job. The pieces of jobs that are processed without preemption in PS(i) are called *job parts for customer i* or simply *job parts* if the choice of a customer is clear from the context or unimportant. A job of customer i that is preempted k times in PS(i) has $k + 1$ job parts.

Now we present Algorithm GS (GS stands for a "global schedule") that obtains a full schedule for Problem M; this full schedule will be denoted by S_A. At any instant, each job will have a status of an *ordinary job* or an *extra job*. Initially, all jobs are ordinary. Some jobs become extra jobs as time goes. An extra job never becomes ordinary again. Job parts have the same status as the corresponding jobs. The idea of the algorithm is as follows. In the single-customer case, if only jobs of a customer i are present, the pseudo-schedule PS(i) obtained by Algorithm PS(i) has total cost that is at most twice the total cost of an optimal off-line solution [2]. This motivates us to try to build a full schedule for the multi-customer case from the pseudo-schedules PS(i), $i = 1, \ldots, m$. If the pseudo-schedules do not overlap (that is, if at any instant, only in at most one of the pseudo-schedules a job is being processed), then clearly the full schedule obtained by superimposing the pseudo-schedules for all customers will also have a total cost which is at most twice the total cost of an optimum off-line solution. The difficulty occurs when pseudo-schedules overlap, because resolving conflicts will inevitably delay the completion of some jobs. In resolving the conflicts, we will make sure that: (1) job parts are not preempted, (2) job parts of a customer are processed in the same order as in the pseudo-schedule for this customer, and (3) the server cannot be idle if there are available jobs. We will schedule deliveries to each customer at all trigger instants for this customer; these deliveries will be called *ordinary deliveries*. An ordinary delivery for a customer i may not be able to take all jobs that are taken in the corresponding delivery of PS(i), because some of these jobs may still be unfinished in S_A; such jobs of customer i get the status of "extra jobs" immediately after this ordinary delivery, and their unfinished job parts form a batch (called *ej-batch*, "ej" standing for "extra jobs"). Each ej-batch is processed contiguously without interruptions at a time chosen by the algorithm, and immediately after this all jobs from the ej-batch are delivered in a special *extra delivery*. Thus, S_A may have twice as many deliveries for customer i as PS(i). Below is a description of Algorithm GS as a set of rules that define the actions of the server at any instant. We assume that at its release time, each job has the status of an ordinary job; the server is idle when there are no available jobs, and cannot be idle otherwise. The server is considered idle before time 0.

Algorithm GS. *Rule* 1. Processing of a job part is never preempted. Job parts of customer i are processed in the same order as in PS(i).

Rule 2. At each trigger instant $\tau > 0$ for a customer i, an ordinary delivery is made for this customer that delivers all already finished undelivered jobs from $J(\tau, i)$. After this, all remaining jobs from $J(\tau, i)$ become extra jobs; their unfinished job parts form an ej-batch.

Rule 3. If processing of an ej-batch has started, it continues without interruptions until the ej-batch is finished. As soon as an ej-batch is finished, an extra delivery is made for the corresponding customer that delivers all jobs of the ej-batch.