mathematics

MDPI

Article

Fast Algorithms for Basic Supply Chain Scheduling Problems

Nodari Vakhania [1,*] and Badri Mamporia [2]

[1] Department of Computer Sciences, Universidad Autónoma del Estado de Morelos, Cuernavaca 62209, Morelos, Mexico
[2] Muskhelishvili Institute of Computational Mathematics, Georgian Technical University, 0159 Tbilisi, Georgia; badrimamporia@yahoo.com
* Correspondence: nodari@uaem.mx

Received: 31 August 2020; Accepted: 1 October 2020; Published: 2 November 2020

Abstract: A basic supply chain scheduling problem in which the orders released over time are to be delivered into the batches with unlimited capacity is considered. The delivery of each batch has a fixed cost D, whereas any order delivered after its release time yields an additional delay cost equal to the waiting time of that order in the system. The objective is to minimize the total delivery cost of the batches plus the total delay cost of the orders. A new algorithmic framework is proposed based on which fast algorithms for the solution of this problem are built. The framework can be extended to more general supply chain scheduling models and is based on a theoretical study of some useful properties of the offline version of the problem. An online scenario is considered as well, when at each assignment (order release) time the information on the next order released within the following T time units is known but no information on the orders that might be released after that time is known. For the online setting, it is shown that there is no benefit in waiting for more than D time units for incoming orders, i.e., potentially beneficial values for T are $0 < T < D$, and three linear-time algorithms are proposed, which are optimal for both the offline and the online cases when $T \geq D$. For the case $0 < T < D$ an important real-life scenario is studied. It addresses a typical situation when the same number of orders are released at each order release time and these times are evenly distributed within the scheduling horizon. An optimal algorithm which runs much faster than earlier known algorithms is proposed.

Keywords: supply chain scheduling; algorithm; batch; release time; delivery; time complexity

1. Introduction

Coordination between different stages in a supply chain is an important issue affecting the overall efficiency of the manufacturing process. Poor coordination of the decisions taken at different stages in supply chain, including scheduling, batching and delivery stages, may result in a poor overall performance [1]. In a supply chain scheduling problem, the batching and delivery activities are to be combined. The orders are released by a supplier to a manufacturer over time, and once processed, they need to be delivered to a customer. To minimize the overall costs, two or more orders can be delivered together in a batch. For example, in some applications, a number of products can be processed by a single batch machine simultaneously (e.g., burn-in operations in semiconductor manufacturing, see Lee and Martin-Vega [2]), or they can be grouped and delivered into a single batch (e.g., in multistage flow shops where the products grouped into batches can be transported between the machines). A batch, formed in accordance with given restrictions, is processed by a batch machine, which can handle a number of jobs or orders simultaneously, in contrast to a machine in traditional scheduling problems that can process at most one job at a time. In the capacitated batch

scheduling problems only a constant number of orders may simultaneously be handled by a batch machine, unlike the uncapacitated version in which a batch machine may handle unlimited number of jobs. Such an assumption is realistic in practice, for example, for vehicles of large capacity when the size of the completed orders is negligible (these can be integrated chips). In the commonly studied models, either the number of batch machines is unlimited or there is a single batch machine or there are two or more parallel batch machines. In the first case, with a sufficient number of batch machines, each batch is completely specified by a set of the orders and by its starting/delivery time. If there is a single batch machine, then each next batch may be assigned to the machine only when it becomes idle, i.e., it completes the delivery of the last batch assigned to it. The case with parallel batching machines is an extension of the previous model in which, in addition, each batch is to be assigned to a particular batch machine.

Overall efficiency of scheduling, batching and delivery decisions in supply chain scheduling problems can be measured in different ways. Here, the measure introduced by Hall and Potts [1] is considered. It combines two criteria in one objective function, the total batch delivery cost and a scheduling cost, which is determined by the total flow time of the orders before their delivery. There is a sufficient number of batch machines and batch delivery time is a constant D. The delivery cost does not depend on the number of the orders in the batch so that there is no restriction on the number of orders that can be assigned to a single batch. The orders are released by a supplier to the manufacturer over time and are to be delivered to a customer in batches. One or more orders can simultaneously be released by the supplier. In this model the processing time of each order is negligible and the emphasis is given on the batching and delivery activities with the aim to minimize the total flow time and the delivery cost of all the released orders. In Figure 1 the problem is represented schematically. A supplier releases the orders over time to a manufacturer. The manufacturer forms batches from the already released orders and delvers them to a customer (there is a single customer and a single manufacturer). Note that these activities are present in nearly all the supply chain scheduling problems, therefore it is of a primary importance to arrange them in a most efficient possible way.

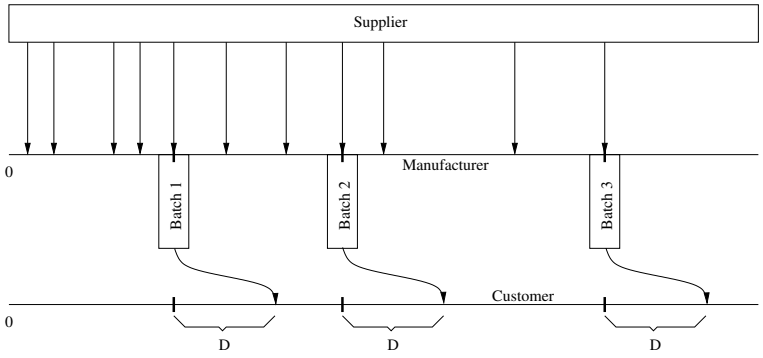

Figure 1. A schematic of the problem.

The purpose of this work is the development of an algorithmic framework leading to polynomial-time algorithms for the above basic supply chain scheduling problem which run faster than the existing ones. The proposed framework gives a new insight to more complex supply chain scheduling problems and can be extended for the solution of these problems as well (see Sections 5 and 6). Hall and Potts [1] have studied wide class of the related general supply chain scheduling problems allowing two or more suppliers, manufacturers and customers. The time complexity of the dynamic programming algorithms that they propose is polynomial in n with the degree of the polynomial being dependent on the number of suppliers, manufacturers and customers. In a model from [1] that fits our setting there are H customers and a single batch for each customer is created (Section 3.1 in [1]). The proposed algorithm can be adopted to the basic setting with a single customer but multiple deliveries (in [1] a single delivery for each customer is accomplished); with H deliveries, the resultant time complexity is $O(n^H)$ (hence the use of this algorithm for the basic problem

would not be beneficial). More recently, Averback and Baysan [3] have studied a model, similar to ours but with a restriction that only one order is released at each order release time. The authors present an algorithm that also uses dynamic programming, verifying basically all possible ways of splitting the solution into the batches, with overall time complexity $O(n^2)$. This dynamic programming method can easily be adopted to our setting (which allows two or more simultaneously release orders) yielding an $O(n^3)$ time complexity. The authors in [3] also consider the online setting, in which, by each order release (assignment) time t all orders released within the interval $(t, t+T]$, $0 < T < \infty$, are known. However, here it is shown that it suffices to consider only the Ts with $0 < T < D$. At the same time, for larger values of T an optimal linear time algorithm is proposed which is better than the earlier known $O(n^3)$ algorithm (see Section 2.2).

Remind that the time efficiency of a computational algorithm for a given problem P is measured in terms of its (worst-case) time complexity [4], which is a function representing the maximum number of elementary operations that the algorithm will require for the generation of a solution to any instance of problem P, where the argument of this function is the length of this instance (the number of computer words used in the representation of this instance, stored in the binary encoding in computer memory). Polynomial-time algorithms (ones with polynomial time complexity) are the most time efficient ones, fastest polynomial-time algorithms are those with lower degree polynomials in their time complexity expression. $n+1$ computer words are required to represent an instance of our supply chain scheduling problem, where n is the total number of orders (each order has a single parameter, its release time, and we need to represent one additional parameter, the delivery cost D). Hence, $O(n)$ is the length of the input in the time complexity expressions of the above mentioned algorithms.

Besides the offline setting, the online setting of the basic supply chain scheduling problem is considered, similar to that from the above mentioned reference [3]: at each assignment (an order release) time t, the information on the next order released within the following T time units, i.e., within the interval $(t, t+T]$, is known, where T is a constant. Our setting is similar to but is less restrictive than that from [3] which assumes that by time t all the orders released within the interval $(t, t+T]$, $0 < T < \infty$, are known. It is shown that there is no benefit in waiting for more than $D-1$ time units at any assignment time in the online setting, i.e., potentially beneficial values for T are $1, 2, \ldots, D-1$, and propose three linear-time offline algorithms which also work for the online case with $T \geq D$. These algorithms are optimal for both the offline and the above online cases for $T \geq D$. For the case $0 < T < D$, an important real-life scenario with the so-called homogeneously released orders is studied. This problem naturally arises in many industries and hence has an independent practical significance. It addresses a typical situation when the same number ν of orders simultaneously become ready for the delivery and the manufacturer repeatedly releases the orders within the same fixed time lapse Δ. As a practical example that fits this setting, consider a food manufacturing process where a supplier dispatches row products to the manufacturer twice a day, say at 8 a.m. (for the day shift) and at 8 p.m. (for the night shift). These products, after processing, are delivered to customers (food distributors or supermarkets). Although the delivery times are flexible, the manufacturer wishes to reduce the cost by delivering the finished products in batches.

In more formal terms, ν orders are released at each order release time and these times are evenly distributed, being separated between each other by magnitude Δ, which is the distance between each neighboring pair of release times in the scheduling horizon. It is shown that for the proposed model, the total delay of the orders can be calculated with an almost constant cost, and, as a consequence, we arrive at a very fast algorithm that optimally solves this problem in $(r^{\max}/k)\log(\lceil r^{\max}/\mu \rceil - \lfloor r^{\max}/\kappa \rfloor)$ steps, where $r^{\max}$ is the maximum order release time (in the offline setting) and μ and κ are functions of ν and Δ. Here, $(r^{\max}/k)$ represents the number of deliveries (k is a fraction of $r^{\max}$). Observe that for the construction of a feasible solution with $(r^{\max}/k)$ deliveries (which is the number of deliveries in an optimal solution), at least $(r^{\max}/k)$ steps are required, and hence the proposed algorithm has "an almost" best possible running time that gives obvious potential tangible

benefits. An implementation of the algorithm may also yield intangible benefits for the workers of the manufacturing production due to reduced delivery costs that may give them a sense of the satisfaction.

The paper is organized as follows. In the next section, the studied supply chain scheduling problem is described and a short overview of some related work is given. In Section 3, basic concepts, definitions and properties are introduced based on which three algorithms are proposed. The sufficient conditions when they construct an optimal solution are given. In Section 4, a practical setting with the homogeneously released orders is considered and a fast optimal solution method is presented. In Section 5, it is shown how the proposed framework can be adopted to more extended models considering, as an example, a two-stage supply chain scheduling model. In Section 6, final comparative analysis and remarks are given.

2. Problem Formulation and Some Related Work

2.1. Problem Formulation

This section starts with a more detailed description of the basic supply chain scheduling problem followed by an overview of some more related work. The proposed model can be described as follows. The orders released over time are to be delivered into the batches. Each formed batch is assigned to a batch machine which may handle any number of orders. A batch (batch machine) is said to be started at time t if the batch machine starts the delivery of all the orders assigned to that batch at that time. The delivery completes at time $t + D$, where D is the cost (time) of the delivery of the batch (this cost is the same for any batch). Order j is released at time r_j, the time moment when this order becomes ready for the delivery to the customer. An already released order j can immediately be delivered, and it may also be delivered at a later time moment. In the latter case, there is a penalty for the late delivery of order j: if order j it is delivered at time t then its delay is $t - r_j$. Thus the total cost of the delivery of each batch is the cost D plus the total delay of the orders assigned to that batch. A (feasible) solution S assigns each order to some batch and delivers all the created batches. The total cost $TC(S)$ of feasible solution S is the sum of the cost of the delivery of all batches that it creates. The objective is to find an optimal solution, one with the minimum total cost. As we will describe later, minimizing the total delivery cost and minimizing the total order delay are contradictory objectives. Hence the problem can alternatively be seen as a bi-criteria optimization problem.

2.2. Related Work

Below, a few survey papers on the subject and some other relevant works are briefly mentioned, and an overview of some recent lines of research is provided.

2.2.1. Early Works and Surveys

In an early survey paper on batch scheduling problems, Webster and Baker [5] classify two branches of batch scheduling problems. They consider the batch availability setting where jobs are processed by a batch machine and can be delivered to a customer once the entire batch is proceed (here a batch machine is not considered as a transportation means). In an alternative job availability setting, a job becomes available for the delivery by a batch machine once its processing is complete. Potts and Kovalyov [6] review the state of the art of batch scheduling problems from the computational complexity point of view. In particular, they classify the batch scheduling problems into the polynomially and pseudo-polynomially solvable and NP-hard ones (see Garey Johnson [4]). They also emphasize that dynamic programming methods are particularly applicable to these problems. Hall and Potts [1] give an extensive overview of the supply chain problems and focus on the integrated batching, scheduling and delivering decisions proposing dynamic programming methods for a number of integrated supply chain scheduling problems with different objective functions. They consider four-stage supply chain scheduling models in which the second and the fourth stages are batch delivery stages. Prior to each delivery stage, the jobs are performed by a single machine at the first

and the third stages. At the first stage, a supplier processes the jobs on a single machine which are then delivered to one or more manufacturers at the second stage. The manufacturers process these jobs on the other machine at the third stage, and then these jobs are delivered to customers at the fourth stage. We have mentioned briefly in the introduction that the dynamic programming algorithm proposed by the authors have polynomial time complexity, although the degree of the polynomial depends on the number of deliveries. The number of deliveries, in turn are limited by the number of manufacturers and customers since a single delivery for each manufacturer and each customer is accomplished. The authors continue their study in the following work [7] considering also parallel machine environment. Chen [8] and Wang et al. [9] overview more recent study and establish computational complexity of a number of integrated scheduling, batching and delivery supply chain scheduling models.

A basic supply chain scheduling problem from an earlier mentioned reference [3] is a restriction of our model in which at each order release time only one order is released. In the online setting that the authors consider, by each assignment time t all orders released within the interval $(t, t+T]$, $0 < T < \infty$, are known (recall that our results imply that it suffices to consider only the Ts with $0 < T < D$), and an $O(n^2 \log n)$ online algorithm is suggested. The competitive ratio ρ of that algorithm (the ratio of the objective value of the solution delivered by the algorithm to the objective value of an optimal offline solution) is $\frac{2D+0.5T}{D+0.5T}$ if $0 < T < 2D$, and it is $\frac{T+D}{T}$ if $T \geq 2D$. The authors also prove that $\frac{2D}{T+D}$ is the best possible competitive ratio of any deterministic algorithm for the setting.

2.2.2. Two-Stage Models with Traditional and Batching Machines

Now some work on more general, two-stage models with both, traditional and a batching machines are addressed. Zhong and Jiang [10] consider such a model with unlimited number of capacitated vehicles. As in the basic model, the orders are released over time by the supplier, which are to be processes first on the machine and then delivered into the batched to the customer. Two different objective functions are considered: to minimize the maximum order dispatching time plus the total delivery cost, and to minimize the sum of the dispatching times of the orders plus the total delivery cost. An exact polynomial-time and heuristic algorithms are proposed for the first and the second objective functions, respectively. Cheng et al. [11] consider another two-stage batch scheduling problem with the objective to minimize the total batch delivery cost plus the total job earliness. Yang [12] considers a similar model in which batch delivery times are priory given. It is an interesting question whether the framework that we propose here can be adopted for the solution of the two-stage supply chain scheduling problems with the above objective functions (see Section 5). Lee and Chen [13] consider single-machine two-stage supply chain scheduling problem in which all jobs are released simultaneously at time 0, are to be processed by a single machine and then delivered to a customer by a batch machine. Models in which batching machines have unlimited and limited capacity are considered and polynomial algorithms for some of these models are proposed. The authors also consider multistage shop scheduling models in which the jobs processed on a machine are to be transported to the next machine by a batch machine. Polynomial and pseudo-polynomial algorithms for some special cases of the problem are proposed. A single-machine two-stage model, similar to one from [13] but with non-simultaneously released jobs is considered by Lu and Zhang [14] (non-simultaneously released orders make the problem significantly more complicated). The batch machine has a limited capacity (there is a single batch machine), there is a single customer and the delivery cost is a constant. The objective is to minimize the maximum order completion time, where an order completion time is the time moment when the batch containing that order reaches the customer. Note that in this commonly considered objective criterion the total batch delivery time is unimportant, which makes, in general, problems with this criterion easier to treat compared to the problems with the objective criterion that we consider here (as there are no two contradictory objective criteria in the former objective function). The authors propose a polynomial-time approximation scheme for the

problem (a family of algorithms that provide an approximation solution in polynomial time with any fixed approximation ratio).

Averback and Baysan [15] extend the one-stage online model from their earlier mentioned work [3] to a two-stage fully online case, the online setting in which no restriction is imposed by constant T, where preemption of an order already scheduled on the machine is allowed. A polynomial-time algorithm with the best possible competitive ratio 2 is proposed. It is an open question whether there exists a direct combinatorial algorithm with a low degree polynomial running time for the offline setting of this model. Another two-stage online model is considered in Ng and Lu [16]. The jobs are again released over time. The processing time of a job on the machine becomes known once this job is released. Once a job completes on the machine, it can be delivered (together with other completed jobs) to the customer by the batch machine (a vehicle). The batch delivery cost is a constant (as in our model) and the batch machine has a limited capacity (unlike our model). The delivery completion time of a job includes its processing time on the machine and also its delivery time by the batch machine. The objective is to minimize the maximum job delivery completion time. The paper presents two online algorithms with the best possible competitive ratios for the preemptive and non-preemptive setting respectively.

Zhong and Chen [17] consider a hybrid two-stage supply chain scheduling problem that involves, both, the scheduling and the transportation phases. Such problems arise in two-stage production: every job, once processed by a machine in the first stage, is to be delivered to the location where the second stage is carried out. The means of transportation is a batch machine. In scheduling theory, problems that involve several-stage production process are called shop scheduling problems. In the flow-shop scheduling problems the processing order or jobs on all stages is the same. This paper considers two-stage flow-shop scheduling environment. In the first problem there is a single machine available at stage 1, and there is a single batch machine, with a limited capacity, to deliver the completed jobs from stage 1 to stage 2 (every job, besides its processing time is characterized by its size, taken into account while a batch of jobs is formed for the transportation). Thus, whenever there are two or more jobs waiting for a batch machine at stage 1, a packing problem of these jobs into the batch machine with a given limited capacity arises. At stage 2 there is also a single machine that has to process the jobs delivered by the batch machine from stage 1. In the second problem there are two parallel alternative machines available at stage 1, whereas the batch machine may deliver only one job at a time. In both problems it is assumed that there is enough buffer space at stage 1 for the completed jobs waiting for the batch machine. The travel time of the batch machine from stage 1 to stage 2 is assumed to be more than that from stage 2 to stage 1 (as the first travel involves the loading and unloading operations for every job). In both problems the objective is to minimize the makespan, i.e., the maximum job completion time at stage 2. It is easy to see that both problems are NP-hard. The authors propose a 2-approximation heuristic algorithm for both problems, i.e., an algorithm that delivers a solution, at most twice worse than an optimal one. Both algorithms adopt and combine known heuristic algorithms for the related two-stage flow-shop and bin packing problems.

2.2.3. Models with Parallel Batch Machines

The settings with parallel batch machines and in which the orders have additional parameters have also been studied. Condotta et al. [18] consider a parallel batch scheduling model in which the processing requirement of any job on any batch machine is the same magnitude p. Besides the release time, every job has also its due-date. Every batch machine has a fixed capacity, and the processing time of a batch is p. The completion time of a job assigned to some batch is starting time of that batch plus the processing time of that batch p (so all jobs assigned to a batch machine complete simultaneously). Two settings are considered. The first feasibility problem seeks for a solution in which all jobs complete by their due dates, and in the minimization version the maximum job lateness (the difference between job completion time and its due date) is to be minimized. It is shown that both problems can be solved in polynomial time. The proposed algorithms use the earlier known technique for scheduling

problems with job release times and due dates: while constructing a feasible schedule, no batch is allowed to be started within some specially declared intervals. As a result, more urgent jobs might not be delayed by less urgent ones, that yields smaller lateness for the urgent jobs. Note that, in this model, unlike our model, the batch machines actually process the orders, and instead of minimizing the sum of the delays the maximum delay is minimized. This model can be extended in a two-stage supply chain scheduling problem with parallel batch machines.

Another parallel batching model again with identical batch machines in the online setting is considered in Li and Yuan [19]. The set of orders is partitioned into the families so that each machine may process simultaneously only jobs from the same family. The online feature of the problem is that the jobs are released over time, i.e., job release times are not known in advance. All jobs have equal processing times, a deadline and a weight. The objective is to maximize the weighted throughout, i.e., the weighted number of the early jobs, ones that complete before their deadlines (the jobs which do not complete by their deadlines are discarded). In this online model, job restarts are allowed; that is, the processing of an already scheduled job might be interrupted, that job being (temporarily) removed from the system, in favor of some other just released job. The already elapsed processing time of the former interrupted job does not count (it is just wasted). The paper considers two special cases of the above problem with two and three parallel identical batch machines. For the first version it is shown that no online algorithm may have the competitive ratio less than 2. Then, an on-line algorithm with the competitive ratio of 3 is proposed for the capacitated version. Though this online algorithm assumes that the time moment when the latest job is released is known in advance. For the special case with 3 identical batch machines, online algorithm with the competitive ratio of approximately 5.2 is presented for the uncapacitated model.

In the model considered by Li [20] there are n jobs to be processed by m parallel batch machines. In general, all jobs can be processed by any machine, however, there are capacity limitations. Machine i has a limited capacity K_i so that the total size of the jobs assigned to that machine as a single batch cannot exceed K_i, whereas besides size s_j, every job j is characterized again by its release time r_j and the processing time p_j. In this setting, it is assumed that every job fits in the machine with largest capacity, however, it may not fit in some other machine, i.e., its size can be more than the capacity of that machine. The processing time of a batch assigned to a machine is the processing time of the longest job (one, with the maximum processing time) from that batch. Every job from the same batch has a common starting and completion times, dictated by the completion time of the previous batch assigned to the same machine (initially, it is 0). The objective is to find a feasible schedule, i.e., assignment of jobs to the batches and assignment of every formed batch to some batch machine satisfying the above constraints, in which the maximum job completion time is the minimum possible. We may note that the problem can be easily solved if there are no capacity constraints and no release times: just form a single batch including all the jobs and process it on any batch machine. The objective value will be then the maximum job processing time, which is obviously a lower bound on the optimal objective value. Otherwise, a special care is to be taken on how to distribute the jobs into the batches and assign these batches to the machines so that to minimize the maximum job completion time. This task becomes difficult, it is easily seen to be strongly NP-hard. For instance, if all jobs have equal processing times and there is only one batch machine, then the problem becomes a version of the bin packing problem with equal bin capacities, which is strongly NP-hard even without job release times (by minimizing the number of batches/bins, the maximum job completion time will obviously be minimized). A few special cases of the described generic problem are addressed. A 5-approximation polynomial-time $O(\max\{m, n^2\})$ algorithm is proposed for the version without job release times, and $(2+\epsilon)$-approximation algorithms are suggested for the case without release times and for the case with arbitrary release times but with a fixed number of different batch capacities; here ϵ is an arbitrarily small positive number. The earlier known algorithms are also adopted to some other special cases yielding some guaranteed approximations.

Xu and Bean [21] consider a batch scheduling problem on unrelated parallel processors that models manufacturing process in the semiconductor industry. Here, one or up to a constant number K orders can be simultaneously processed by a batch machine (earlier mentioned burn-in ovens in the semiconductor production that may handle several boards with a number of chips). Each job (a board) gets released over time (at its release time) and has a weight (reflecting somehow its priority in the production system). The problem is to distribute jobs into batches of size at most K and determine the formed batch sequences on the unrelated machines so as to minimize the total weighted tardiness (the tardiness of a job is determined as the difference between its completion and release times (if job completes earlier than its release time then the tardiness is 0). The problem is formulated as nonlinear integer programming model and is shown that this (primal) problem can be solved by a corresponding dual problem with a nonlinear relaxation. Since these problems are NP-hard, a two heuristic genetic algorithms are proposed. Based on the results of the computational experiments, these algorithms show a favorable practical performance.

2.2.4. Recent Development

As with some more recent developments in supply chain scheduling, the settings with learning and deterioration effects are dealt with in Fan et al. [22]. By the deterioration effect, the processing time of an order depends on its starting time, the later it starts, the more processor time it needs (due to the fact that the raw materials that need to be processed deteriorate over time). On the other hand, by the learning effect, the processing time of an order depends also on the position of this order among the other orders, the higher is this position, the smaller is the amount of time that this order will need to complete: the workers and the machines can improve the production efficiency with more processing experiences. These effects are reflected on order processing times through the corresponding auxiliary parameters. This paper, in particular, considers a scheduling problem in which the orders are partitioned into different groups, and the orders of each group are to be processed one after another in a serial way by a serial-batch machine, whereas the completion time of all the orders from the batch is the completion time of that batch (no matter at what time moment an order from the group is assigned to the batch, all of them will complete at the same time). The number of orders of the same group that can be assigned to a single batch machine is restricted by the batch capacity, which is the same constant for all batch machines. The processing time of every job is schedule-dependent and is determined on the basis of the combined deterioration and learning effects. Each group and batch machine requires a setup time, which occurs when one group/batch machine is followed by another group/batch machine. In addition, each group is characterized by its release time (before which no job of this group can be started). The objective is to minimize the maximum job (batch) completion time, the so-called makespan. First, some properties of an optimal schedule for the version without group release times are studied. Then some scheduling rules and a polynomial-time algorithm using the proposed rules are described. Finally, an evolutionary algorithm for the general model is described and its practical performance is analyzed. In a recent work, Hsu and Wang [23] emphasize the importance of a proper resource allocation in complex supply chain problems and propose a number of metaheuristic algorithms to optimize resource allocation. Another recent paper [24] studies the effect of import, export and investment issues on the economic grow in eastern European market.

3. Methodology

3.1. Materials

In this section, first, the general scheduling strategy that the proposed algorithms apply is described, and then the relevant definitions are introduced. Similar scheduling strategies can be applied to different supply chain scheduling problems (see Section 5). Recall that in these problems, a supplier releases the orders over time. We temporarily assign the released over time orders to the so-called pending batch. For example, in Figure 1, the orders released at the first five release times are

temporarily assigned to the corresponding pending batches (correspondingly, five different pending batches are successively formed, where each newly formed one extends the previous pending batch). Once a decision is made to deliver the orders from the (last) bending batch all together in Batch 1, these orders are assigned to that batch and are dispatched; the latter pending batch disappears and the orders released at the next order release time are assigned to a new pending batch. Note that since the orders from a pending batch are not delivered, a pending batch is not a normal batch: neither the delivery time of the orders assigned to that batch nor the subset(s) of the orders from that batch which will simultaneously be delivered in a single batch is defined.

At every decision point t (which is an order release time), there is a set of the already delivered orders and a set of the pending orders, ones from the current pending batch B^t. We denote by $r^1, r^2, \ldots, r^k$, $r^1 < r^2 < \cdots < r^k$, all the distinct release times of the pending orders from batch B^t (here $t = r^k$). We shall refer to each r^i as a (potential) splitting point by time t. We denote by $r^{\max}$ the last splitting point (which is the maximum order release time in the off-line setting).

At splitting point r^k the current pending batch $B^{r^{k-1}}$ is said to be extended if the new pending batch B^{r^k} is formed in which the orders of batch $B^{r^{k-1}}$ and the orders released at time r^k are included.

Alternatively, pending batch $B^{r^{k-1}}$ is said to be split at point r^l, $1 \le l \le k-1$, if the orders released by time r^l are delivered in a (normal) batch at time r^l and the orders released after that time up to (including) time r^k are assigned to the new pending batch B^{r^k}. So a splitting point is a time moment at which the current pending batch may potentially split into two distinct batches: the first of these batches is a normal newly created batch and the second one is the new pending batch.

If there occurs the actual splitting of pending batch $B^{r^{k-1}}$ at some point r^l (i.e., the algorithm under the consideration takes such decision), then that point is said to become active at time r^k. In this case we split pending batch $B^{r^{k-1}}$ into two batches: the first one is a normal batch containing all the yet undelivered orders released by time r^l and is delivered at time r^l, and the second one is a new pending batch B^{r^k} containing the remaining orders from batch $B^{r^{k-1}}$ (ones released after time r^l) and the orders released at time r^k.

Lemma 1. *Every batch in an optimal solution S_{opt} is delivered at some splitting point and all the orders released at that point are delivered in the same batch.*

Proof. Let B be a batch in solution S_{opt}, and r be the maximum release time of an order from that batch. Batch B cannot be delivered before time r, and if it is delivered after time r then the delivery cost can obviously be decreased by delivering this batch at time r. In particular, all the orders released at time r must be delivered in batch B as the total delay of these orders will then be 0. □

Let $\tau_i = [r^i, r^{i+1}]$, where we will use $|\tau_i| = r^{i+1} - r^i$ for the length of interval τ_i, $i = 1, 2, \ldots, k$. Let, further, $\nu(r^i)$ ($\nu'(r^l)$, respectively) be the number of the orders in pending batch B^t having the release time no larger than r^i (having the release time r^l, respectively), and let $\nu^+_{r^l}(r^i)$ be the number of the orders in pending batch B^t released between times r^l and r^i including ones released at these times; here $i = 1, 2, \ldots, k$, $t = r^k$, and in the latter case, $l < i$.

$$TD(r^1, r^k) = \sum_{i=1}^{k-1} \nu(r^i)|\tau_i| = \sum_{i=1}^{k-1} \nu'(r^i)(r^k - r^i) \tag{1}$$

is the total delay of the orders from the pending batch B^t if they are delivered in a single batch at time r^k. More generally, if we consider only the orders released from time r^l, then

$$TD(r^l, r^k) = \sum_{i=l}^{k-1} \nu^+_{r^l}(r^i)|\tau_i| = \sum_{i=l}^{k-1} \nu'(r^i)(r^k - r^i) \tag{2}$$

will be the total delay of the orders from pending batch B^t released between time moments r^l and r^k (including these time moments) if they are delivered in a single batch at time r^k (note that for such deliveries the total delay of the orders released at time r^k is 0).

Clearly, the last delivery occurs at the release time $r_{\max}$ of the latest released order in (off-line) optimal solution S_{opt}. Besides this active splitting point in solution S_{opt}, we may have other predictable active splitting points. A splitting point r^i is called terminal if either $r^i = r^{\max}$ (in the offline setting) or

$$|\tau_i| \geq D/\nu'(r^i). \tag{3}$$

Lemma 2. *If r^i is a terminal splitting point then there is a batch that delivers the orders released at time r^i at time r^i in an optimal solution S_{opt} (this batch may also contain orders released before time r^i).*

Proof. By Lemma 1 all the orders released at some splitting point are to be delivered together in one batch. By contradiction, suppose the $\nu'(r^i)$ orders released at time r^i are delivered at time r^{i+1} or later. Since r^i is a terminal point, the total delay of these orders will be no less than D (whereas it will be 0 if these orders are delivered at time r^i). The lemma obviously follows since the cost of the delivery of a batch at time r^i containing all the orders release at time r^i is D. □

As a simple consequence of the above lemma, we obtain that for a problem instance in which the distance between any two neighboring splitting points is at least D, a single batch in solution S_{opt} delivers all the orders released at each splitting point, and this applies to both, offline and online settings.

Corollary 1. *In the online version of the problem, there is no benefit in waiting for more than D time units for the next incoming order. Hence, for any $T \geq D$, the online setting is reduced to the offline case. Therefore, the potentially interesting values for T in the online setting are ones that fall in the interval $[0, D)$.*

Proof. Immediately follows from Lemma 2 (which assures that if the distance from splitting point r to the next splitting pint r' is at least D, then all orders released at time r can be delivered at that time). □

3.2. Methods

In this section, we describe our algorithms and give the conditions when they provide an optimal solution to our problem.

Algorithm 0. According to Lemma 2, our first *Algorithm 0* forms our initial solution σ as follows. Iteratively, at each splitting point r^i, it extends the pending batch $B^{r^{i-1}}$ with the orders released at time r^i forming in this way the new pending batch B^{r^i} unless r^i is a terminal splitting point. In the latter case the algorithm creates a batch that delivers at time r^i the orders from pending batch $B^{r^{i-1}}$ and the orders released at time r^i. In this way, Algorithm 0 naturally partitions the incoming orders into the segments, with each segment the corresponding terminal point being associated. For that segment, Algorithm 0 creates a single batch (the segment batch) delivering all orders released within that segment at the corresponding terminal splitting point. Since a segment can be solutiond independently from the other segments, from here on, we concentrate our attention to a single segment to which we shall refer to as the segment and will denote by $r^1, \ldots, r^k$ the splitting points from that segment (r^k being the terminal splitting point of that segment).

Lemma 3.

(i) Algorithm 0 has a best possible time complexity of $\Theta(r^{\max})$.
(ii) Every active splitting point in solution σ is also an active splitting point in an optimal solution S_{opt}.
(iii) Solution σ is optimal if every splitting point is terminal. Otherwise, it remains optimal if for every terminal splitting point r^k in it,

$$TD(r^\iota, r^k) \leq D, \tag{4}$$

where r^ι is the earliest splitting point (the minimum order release time) in the segment containing terminal splitting point r^k.

Proof. Algorithm 0 works on $r^{\max}$ iterations with a constant cost at each of these iterations (the orders released at each splitting point can be included into a batch with a constant cost and condition (3) can also be verified in a constant time at that point). This implies that time complexity of the algorithm is $\Theta(r^{\max})$. It is also a best possible time complexity, as any feasible solution must contain the orders released at each of the $r^{\max}$ splitting points and hence all of them need to be considered in the linear fashion. Claim (ii) and the first statement of claim (iii) immediately follow from the construction of solution σ and Lemma 2.

We show the second statement of claim (iii). Let $r^i \in \{r^1, \ldots, r^{k-1}\}$. Note that inequality (4) essentially says that if the orders released up to point r^{k-1} in the segment containing point r^k in pending batch B^{r^k} are delivered at time r^k, then their total delay will be no greater than D, the cost of the creation of a new separate batch at or before time r^{i-1}. In particular, the total delay of the orders from that segment of pending batch B^{r^k} released between points r^ι and r^i within the interval $[r^i, r^k]$ would also be no greater than D (in case the corresponding new batch is created). Consider a feasible solution σ' obtained from solution σ by delivering a separate batch at point r^i. If that solution is optimal, it delivers a batch also at time r^k (claim (ii)). So in solution σ', the first batch delivers the orders released by time r^i at that time, and second batch delivers the orders released between times r^{i+1} and r^k at time r^k. We have

$$TC(\sigma') = TC(\sigma) - TD_\sigma((r^\iota, r^i) \to [r^i, r^k]) + D \geq TC(\sigma),$$

where $TD((r^\iota, r^i) \to [r^i, r^k])$ stands for the total delay of the orders from pending batch B^{r^k} released between points r^ι and r^i within the interval $[r^i, r^k]$. We have

$$TD((r^\iota, r^i) \to [r^i, r^k]) \leq TD(r^\iota, r^k) \leq D.$$

Thus we showed that the terminal splitting point r^k is the only active splitting point of the corresponding segment of optimal solution S_{opt}. Since this holds for any segment of solution σ, it is optimal. □

From here on in this paper, we assume that none of the optimality condition from (iii) in Lemma 3 is satisfied. Note that since all the active splitting points of the initial solution σ are also active splitting points in an optimal solution S_{opt}, if solution S_{opt} possesses additional active splitting points then it may contain two or more batches for the jobs of some segment.

Algorithm 1. Our next algorithm employs the concept of the relative total delay that we define now. Consider the pending batch containing the orders released between pints r^l and r^{l+k-1} (including these points, for some positive integer $k > 1$). Here, we assume that the orders released before time r^l are already delivered. These orders may be delivered in a single batch at time r^{l+k-1} or alternatively, they can be delivered together with the orders released at the next splitting point r^{l+k} in a single batch at time r^{l+k}. We may compare these alternatives employing the following definition of the relative total delay

$$RTD(r^l, r^{l+k}) = v^+_{r^l}(r^{l+k-1})(r^{l+k} - r^{l+k-1}).$$

Lemma 4. *Suppose k is the minimum integer such that*

$$RTD(r^l, r^k) > D. \tag{5}$$

Then in an optimal solution S^{opt}, the orders released between points r^l and r^{l+k-1} are not delivered at time r^{l+k} (i.e., they are delivered at one or more points from set $\{r^l, \ldots, r^{l+k-1}\}$ in solution S^{opt}).

Proof. By the way of contradiction, suppose the orders released between points r^l and r^{l+k-1} are delivered at time r^{l+k} in solution S^{opt}. Consider an alternative solution S' obtained from S^{opt} by delivering these orders in a single batch at time r^{l+k-1}. It is easily seen that the reduction in the total delay of the latter orders in solution S' (compared to that in solution S^{opt}) is $RTD(r^l, r^{l+k})$, whereas the cost of the newly created batch that dispatches at time r^{l+k-1} in solution S' is D, i.e.,

$$TC(S') = TC(S^{opt}) + D - RTD(r^l, r^{l+k}).$$

Then by condition (5) $TC(S') < TC(S^{opt})$ and hence S^{opt} cannot be an optimal solution. □

Algorithm 1 relies on the above lemma and on Lemma 2. Iteratively, if the next splitting point r^{l+k-1} is terminal, it creates a batch that delivers at that time the orders from pending batch $B^{r^{l+k-2}}$ and the orders released at time r^{l+k-1}. Otherwise, it verifies condition (5): if the condition is not satisfied, the orders released at time r^{l+k-1} are added to the current pending batch; otherwise, a new batch in which these orders (the ones released between points r^l and r^{l+k-1}) are delivered at time r^{l+k-1} is created. Importantly, Algorithm 1, retains the time complexity of Algorithm 0 (this can be seen similarly as in Lemma 3).

Algorithm 2. From Lemma 4, if condition (5) for point r^k is satisfied, the orders released between points r^l and r^{l+k-1} are delivered at one or more splitting points from set $\{r^l, \ldots, r^{l+k-1}\}$ in solution S^{opt}. Algorithm 1 creates a single delivery at point r^{l+k-1}. An optimal solution may contain more than one active splitting points from set $\{r^l, \ldots, r^{l+k-1}\}$. Our next algorithm considers such possibility employing a different condition for creating each next active splitting point. Let r^i be the release time of the earliest released yet undelivered order from the current pending batch B^{r^l}.

Algorithm 2 verifies if the total delay of the (yet undelivered) orders from pending batch B^{r^l} (released between times r^i and r^l) if delivered at time r^l will be less than batch delivery cost D:

$$TD(r^i, r^l) < D. \tag{6}$$

If the condition is satisfied, then it is repeatedly verified for the next splitting point. Otherwise, we call point r^{l-1} (the latest splitting point for which the inequality is satisfied) a semi-terminal splitting point.

Whenever splitting point r^l is a semi-terminal or terminal point in pending batch B^{r^l}, Algorithm 2 creates a new batch that delivers all the orders from batch B^{r^l} at time r^l. Thus, Algorithm 2 creates all the active splitting points that Algorithm 0 creates and it also creates additional splitting points in each segment if the condition (4) in Lemma 3 is not satisfied.

4. Applications

In this section, an important special case of our problem which naturally arises in a number of industries and hence has also the practical significance is studied. First the problem is introduced and then the performance of the proposed algorithm is studied. The special case addresses a typical situation when the same number of orders are released at each splitting point which are evenly separated between each other. We call such orders homogeneously released orders; more formally, there are positive integer constants Δ and ν such that for each splitting point r^i (and r^{i+1})

$$r^{i+1} - r^i = \Delta$$

and

$$\nu'(r^i) = \nu.$$

It is less costly to calculate the total order delay in a feasible solution to this problem. In particular, it is easy to see that the total delay of the orders released at points $r^l, r^{l+1}, \ldots, r^{l+k}$ if they are delivered in a single batch at time r^{l+k} (for a non-negative integer k) will be

$$TD(r^l, r^{l+k}) = \nu\Delta \sum_{i=1}^{k} i = \nu\Delta \frac{k(k+1)}{2}.$$

Likewise, the relative total delay

$$RTD(r^l, r^{l+k}) = \nu^+_{r^l}(r^{l+k-1})(r^{l+k} - r^{l+k-1}) = (k-1)\nu\Delta.$$

Lemma 5. *If for an instance with the homogeneously released orders $\nu\Delta \geq D$ then all the splitting points are terminal. Hence, Algorithm 0 finds an optimal solution for that instance.*

Proof. The first claim easily follows and the second one follows from Lemma 3. □

Suppose now that $\nu\Delta < D$, let

$$\kappa = \max_k TD(r^l, r^{l+k}) = \nu\Delta \frac{k(k+1)}{2} \leq D,$$

and let

$$\mu = \max_k RTD(r^l, r^{l+k}) = (k-1)\nu\Delta \leq D\}.$$

It is easy to see that Algorithm 1, applied to an instance with the homogeneously released orders, will deliver the orders released at points $r^1, \ldots, r^\kappa$ in a single batch at time r^κ, the orders released at points $r^{\kappa+1}, \ldots, r^{2\kappa}$ in a single batch at time $r^{2\kappa}$, and so on, unless one of these points is terminal; then it will deliver all the orders from the current pending batch in a separate batch at that splitting point. Algorithm 2 works similarly—we just replace κ with μ in its description. Notice that the time complexities of Algorithms 1 and 2 are reduced to $\Theta(r^{\max}/\kappa)$ and $\Theta(r^{\max}/\mu)$, respectively.

Since every segment can be dealt with independently from the other segments, without loss of generality we shall concentrate our attention to a single segment with $r^{\max}$ being the maximum order release time in that segment. Note that $\kappa < \mu$ and hence $\lceil r^{\max}/\kappa \rceil > \lceil r^{\max}/\mu \rceil$ for any $k \geq 2$.

Lemma 6. *In optimal solution S^{opt} to an instance with the homogeneously released orders the number of batches is not less than $\lceil r^{\max}/\mu \rceil$ and it is not more than $\lceil r^{\max}/\kappa \rceil$.*

Proof. By the way of contradiction, suppose first that the number of batches in solution S^{opt} (in the segment) is less than $\lceil r^{\max}/\mu \rceil$. Then there must exist two neighboring active splitting points r^{l-1} and r^{l+k} in solution S^{opt} with $k > \mu$. Hence,

$$RTD(r^l, r^{l+k}) = (k-1)\nu\Delta > D.$$

However, then clearly, solution S^{opt} can be improved by creating a new batch that delivers the orders released between points r^l and r^{l+k-1} (including these endpoints) at time r^{l+k-1}, a contradiction.

Suppose now that the number of batches in solution S^{opt} (in the segment) is more than $\lceil r^{\max}/\kappa \rceil$. Then there must exist three neighboring active splitting points r^l, r^{l+k} and $r^{l+k+k'}$ in solution S^{opt} such that $k + k' \leq \kappa$. Hence,

$$TD(r^l, r^{l+k+k'}) = \nu\Delta \frac{(k+k')(k+k'+1)}{2} \leq D. \tag{7}$$

Let S' be the solution obtained from solution S^{opt} by eliminating the batch starting at time r^{l+k} in S^{opt} and delivering at time $r^{l+k+k'}$ the orders from the eliminated batch together with the orders from the batch starting at that time in solution S^{opt}. We have

$$TC(S') = TC(S^{opt}) - D + TD([r^l, r^{l+k}] \to r^{l+k+k'}),$$

where $TD([r^l, r^{l+k}] \to r^{l+k+k'})$ is the total delay of the orders released between times r^l and r^{l+k} within the interval $[r^{l+k}, r^{l+k+k'}]$ in solution S', which is precisely the increase in the total delay of the orders in solution S' compared to solution S^{opt}. Since

$$TD([r^l, r^{l+k}] \to r^{l+k+k'}) < TD(r^l, r^{l+k+k'}) \leq D$$

(the first inequality is obvious and the second one follows from inequality (7)),

$$TC(S') < TC(S^{opt})$$

holds, another contradiction. The lemma is proved. □

Let now consider feasible solutions with a given number β of batches with $r^{\max} = \beta k + \rho$, $0 \leq \rho \leq \beta - 1$. The β-canonical solution $S^C(\beta)$ with β batches is defined as follows: if $\rho = 0$ then repeatedly, each batch from $S^C(\beta)$ delivers the orders released at the next k successive splitting points: the first batch delivers the orders released at times $r^1, \ldots, r^k$ at time r^k, the second batch delivers the orders released at times $r^{k+1}, \ldots, r^{2k}$ at time r^{2k}, and so on, the last batch delivers the orders released at times $r^{(\beta-1)k+1}, \ldots, r^{\beta k}$ at time $r^{\beta k} = r^{\max}$. If $\rho > 0$ then solution $S^C(\beta)$ is defined similarly to case $\rho = 0$ with the only difference that the first $\beta - 1$ batches contain the orders released at the next $k + 1$ successive splitting points and the batch delivery times are modified correspondingly. Note that we can construct β-canonical solution $S^C(\beta)$ similarly as in Algorithm 1 (Algorithm 2) just replacing κ (μ) by k.

Lemma 7. *The β-canonical solution $S^C(\beta)$ to an instance with the homogeneously released orders is constructed in time $\Theta(r^{\max}/k)$ (with $r^{\max} = \beta k + \rho$, $0 \leq \rho \leq \beta - 1$) and has the minimum cost among all feasible solutions with β batches.*

Proof. From $r^{\max} = \beta k + \rho$ we immediately obtain that the cost of the construction of solution $S^C(\beta)$ is $\Theta(r^{\max}/k)$ (similarly to Algorithms 1 and 2). Now we turn to the optimality proof for the case $\rho = 0$. Let us consider two neighboring batches B^i and B^{i+1} delivering at times $r^{(i+1)k}$ and $r^{(i+2)k}$, respectively, the orders released at times $r^{ik+1}, \ldots, r^{(i+1)k}$ and $r^{(i+1)k+1}, \ldots, r^{(i+2)k}$, respectively, in solution $S^C(\beta)$. Consider an alternative solution $S'(\beta)$, which contains the same batches as solution $S^C(\beta)$ except that the sets of orders in the ith and and $(i+1)$st batches are modified: the ith batch contains all the orders from batch B^i except the ones released at time $r^{(i+1)k}$; the latter orders are assigned to the $(i+1)$st batch in solution $S'(\beta)$ together with the orders from batch B^{i+1}. In this way, the delivery time of the first batch is $r^{(i+1)k-1}$ and the starting time of the second one is $r^{(i+1)k}$ in solution $S'(\beta)$. Clearly, the total delay in the ith batch in the latter solution is decreased by $RTD(r^{ik+1}, r^{(i+1)k}) = (k-1)\nu\Delta$ compared to batch B^i, and the total delay in the $(i+1)$st batch in solution $S'(\beta)$ is increased by $k\nu\Delta$ compared to batch B^{i+1}, i.e.,

$$TC(S'(\beta)) = TC(S^C(\beta)) - (k-1)\nu\Delta + k\nu\Delta,$$

and hence $TC(S'(\beta)) > TC(S^C(\beta))$. The repeated application of the same interchange argument easily yields that a feasible solution containing two batches which differ by more than one in the number of the covered splitting points (the release times of the orders from these batches) cannot have a total cost less than $TC(S^C(\beta))$, which proves the lemma for case $\rho = 0$. In case $\rho > 0$, there exists no

solution in which each batch covers the same number of splitting points. Hence in a solution with the minimum cost with β batches, at least two batches must extend to different number of splitting points. The interchange argument applied above for case $\rho = 0$ easily yields that there can be no benefit in creating batches that differ by more than one in the number of the covered splitting points. Hence, $TC(S^C(\beta))$ is the minimum possible total cost for any feasible solution with β batches. □

Corollary 2. *An optimal solution to an instance with the homogeneously released orders can be obtained in the number of steps bounded above by* $(r^{\max}/k)\log(\lceil r^{\max}/\mu\rceil - \lfloor r^{\max}/\kappa\rfloor$.

Proof. Based on Lemmas 6 and 7 we can easy construct an optimal solution combining the procedure for the creation of the β-canonical solution with the binary search. We consider trial values for β from the interval $[\lfloor r^{\max}/\kappa\rfloor, \lceil r^{\max}/\mu\rceil]$ (see Lemma 6) using the binary search: for each β from this interval, we construct the β-canonical solution $S^C(\beta)$ by the above mentioned procedure in time $\Theta(r^{\max}/k)$ (see Lemma 7). The time complexity also follows since the number of the trial βs is bounded above by $\log(\lceil r^{\max}/\mu\rceil - \lfloor r^{\max}/\kappa\rfloor)$. □

5. Extensions

In this section, a possible extension of the studied model is considered and it is shown how the basic notions and properties earlier introduced for our basic model can be adopted for the extended setting. Consider a two-stage supply chain scheduling model, in which we have a traditional machine and the batch machines with the delivery cost D. The processing requirement of order o_i on the machine is p_i. Order o_i cannot be delivered until it is finished on the machine, i.e., it is processed for p_i units of time on the machine once it is already released. A feasible solution specifies the starting time of each order (job) on the machine and the starting time of each batch (that may contain any number of the already finished and yet undelivered orders).

In the offline setting, a feasible solution can be constructed in two stages. At the first scheduling stage all the orders are scheduled on the machine, and at the second delivering stage the scheduled orders are distributed into the batches.

It can be easily observed that none of the two claims from Lemma 1 hold (in general) for an optimal solution S_{opt} for this two-stage supply chain scheduling problem according to our definition of a spitting point. However, if this notion is redefined for the two-stage problem as an order completion time on the machine, then Lemma 1 can be reformulated as follows:

Lemma 8. *Every batch in an optimal solution S_{opt} to the two-stage supply chain scheduling problem is delivered at the completion time of some order and that batch contains all the yet undelivered finished orders.*

Proof. Similarly to Lemma 1, consider a feasible solution containing a batch that is delivered between the completion times of two successively performed orders i and j on the machine. Then the total cost of that solution can clearly be decreased by replacing this batch with another batch that delivers the orders from the former batch at the completion time of order i. Likewise, a batch that does not include all the orders released by the starting time of that batch can be replaced by a batch which contains all these orders without increasing the total cost. □

It is helpful to look at the problem in a two-dimensional scale representing schematically the machine time and the delivery costs by X-axes and Y-axes, respectively. The X-axes can be seen as the already formed sequence of all the orders with their starting times, constructed at the first scheduling stage. It may contain two kinds of points, an order release time and an order completion time; we shall refer to the first and the second type of points, respectively, as the type (1) and the type (2) splitting points, respectively. Note that an active splitting point can only be a type (2) splitting point, and that the same point may be a type (1) and also a type (2) splitting point (e.g., an order may be completed at the release time of another order). At the second delivering stage some type (2) splitting points are

extended by the vertical line segments of the length D (so a complete solution can be represented by extending its every active splitting point with a vertical line segment of the length D).

The X-axes may contain a gap, a machine idle time, which will occur in case all the released orders are already finished on the machine and there is no new order released at the completion time of the latest scheduled order, say i (so a gap is a time interval on the X-axes between the completion time of order i and the starting time of the next released order, in case such an order exists). Note that the left end-point of a gap is a type (1) splitting point and its right end-point is a type (2) splitting point.

A block is a sequence of the orders, the first of which starts immediately after a gap (or at the earliest order release time), there is no gap in between two successive orders and the last scheduled order of the sequence is immediately succeeded by a gap (or there is no more order to be scheduled).

Call a sequence of all the simultaneously released orders on the X-axes a chain; a sub-chain of a chain consists of some successively scheduled orders of that chain. Note that to each (sub)chain C a particular order release time (a type (1) splitting point) corresponds, which will be referred to as the release time of that (sub)chain and denoted by r_C.

Let the offset of (sub)chain C be the difference between the starting time of the earliest scheduled order of this (sub)chain and the release time of that (sub)chain, and let c_C denote the completion time of the last scheduled job of (sub)chain C (a type (2) splitting point).

Property 1. *The X-axes of an optimal solution S_{opt} to the two-stage supply chain scheduling problem is formed by a sequence of (sub)chains. Every block in that solution starts by a (sub)chain with 0 offset and is followed by zero or more (sub)chains with the positive offset. Furthermore, every delivery in that solution occurs at time c_C, for some (sub)chain C (a type (2) spitting point).*

Proof. The first part can be established using a simple interchange argument, and the second part follows from Lemma 8. □

For an already constructed X-axes, consider a (sub)chain $C = (o_1, \ldots, o_k)$ and a type (2) splitting point π, $\pi \geq c_C$ (a potential delivery time of a batch including the orders in C). We can calculate $TD(C, \pi)$, the total delay of the k orders from (sub)chain C if they are delivered at time π, as the sum of the delays of all these k orders:

$$TD(C, \pi) = k(\pi - r_C). \tag{8}$$

Denote by r_B and c_B the starting and completion times, respectively, of block B. Suppose r' and r'' are type (2) splitting points from block B (the interval $[r', r'']$ may include more than one (sub)chain from block B), and let π be another type (2) splitting point with $\pi \geq r''$ (point π not necessarily belongs to the interval of block B). The total delay of all the orders scheduled within the interval $[r', r'']$ in block B if they are delivered at time π can clearly be obtained by summing up the total delays of the (sub)chains from that interval:

$$TD(B, [r', r''], \pi) = \sum_{C \in [r', r'']} TD(C, \pi). \tag{9}$$

Now, point c_B is the terminal splitting point in block B if either $c_B = r^{\max}$ or there exists a splitting point r' such that

$$TD(B, [r', c_B], \pi) - TD(B, [r', c_B], c_B) \geq D, \tag{10}$$

where π is the completion time of the order scheduled the first after time c_B (note that not necessarily a block possesses the terminal splitting point).

Note that the left hand-side of inequality (10) is a natural adaptation of the concept of the relative total delay for the extended model applied to points c_B and π. Below we state a version of Lemma 2 for the extended model.

Lemma 9. *Suppose block B possesses the terminal splitting point c_B. Then there is a batch in solution S_{opt} that delivers (some) orders from block B at time c_B.*

Proof. We can use the dominance argument similar to that from Lemma 2. □

Now a segment is defined as a sequence of blocks ending with a block possessing the terminal splitting point and Algorithm 0 is straightforwardly extended for the two-stage model. Algorithms 1 and 2 can similarly be extended based on the modified notions of total delay and the relative total delay. Note that these algorithms now consist of two stages, the scheduling and the delivering stages.

6. Discussion and Concluding Remarks

A new algorithmic framework that allows to build efficient fast algorithms for the considered basic supply chain scheduling problems was proposed. The conditions under which the proposed algorithms for the basic supply chain scheduling problem provide an optimal solution were established. These algorithms serve as a basis for the construction of a very fast optimal algorithm for the homogeneously released orders. Using similar construction ideas, it might be possible to extend this algorithm for the setting with the non-homogeneously released orders.

As mentioned earlier in the introduction, although the problem was stated as a single-criteria optimization problem, it is essentially a by-criteria problem. Indeed, note that, given a feasible solution S, $TC(S)$ is the sum of the two different magnitudes, the total delivery cost of all batches (which is D multiplied by the number of the batches in that solution) and the total delay of the orders in all the delivered batches. Minimizing the first magnitude will maximize the second one, and visa-versa. Hence the two objectives, minimizing the total number of batches (hence minimizing the total batch delivery cost) and minimizing the total order delay are contradictory, which precisely makes the problem non-trivial. In this paper, the problem was treated as a single-criteria problem but alternatively, different extensions of this problem with the same objective function can be studied as bi-criteria problems. Such a study may reveal new useful properties of these problems and may permit to widen the class of the supply chain scheduling problems allowing other objective criteria. These multi-criteria optimization problems can be approached using the traditional Pareto-optimality or, for example, a recently proposed threshold-optimality measure [25].

In the introduction, the dynamic programming algorithms from [1,3,7] proposed earlier for the relevant supply chain scheduling problems were mentioned. It is convenient to use dynamic programming for such problems because the two contradictory criteria hidden in the objective function can be evaluated together as sum of the two quantities (the total order delay and the total delivery cost) in the corresponding recurrent relations. However, these dynamic programming algorithms yield the enumeration of a considerable number of alternatives which make them not too time efficient. Unlike the dynamic programming algorithms, the proposed here algorithms are direct combinatorial algorithms based on a theoretical study of the structural properties of the problem. This makes these algorithms more time efficient, but at the same time, more difficult to construct because the two contradictory criteria in the objective function, which cannot be dealt with independently from each other. Nevertheless, we have succeeded to construct an optimal algorithm for the case of the homogeneously released orders. Moreover, in previous section, it was shown how the proposed theoretical framework can be extended to a two-stage supply chain scheduling problem, an immediate extension of the studied here model with two types of resources, a traditional machine and a batch machine (e.g., a vehicle). It was shown how the proposed main framework for the basic model can be adopted for a two-stage supply chain scheduling problem. Alternatively, models with two batch machines, the second of which is a transportation vehicle can be considered. These models can further be extended by allowing parallel batch machines. The adaptation of our framework for these models would be somewhat similar. In particular, the studied here basic model can gradually be extended to more complex real-life supply chain scheduling problems and the proposed algorithms can accordingly

be expanded. Efficient solution of the studied basic models might be essential for the development of fast solution methods for more complex real-life supply chain scheduling environments.

Author Contributions: Conceptualization, Methodology N.V.; Formal Analysis, Funding B.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Shota Rustaveli National Science Foundation of Georgia (grant DI-18-1429).

Acknowledgments: The authors are grateful to the anonymous referees, especially to one of them for the extensive constructive comments that have helped to improve the structure and the organization of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hall, N.; Potts, C. Supply chain scheduling: Batching and delivery. *Oper. Res.* **2003**, *51*, 566–584. [CrossRef]
2. Lee, C.Y.; Reha, U.; Louis, A. Martin-Vega. Efficient Algorithms for Scheduling Semiconductor Burn-In Operations. *Oper. Res.* **1992**, *40*, 764–775. [CrossRef]
3. Averback, I.; Maysan, M. Batching and delivery in semi-online distribution systems. *Discret. Appl. Math.* **2013**, *161*, 28–42. [CrossRef]
4. Garey M.R.; Johnson, D.S. *Computers and Intractability Guide to the Theory of NP—Completeness*; Freeman: San Francisco, CA, USA, 1979.
5. Webster S.; Baker, K.R. Scheduling groups of jobs on a single machine. *Oper. Res.* **1993**, *43*, 693–703. [CrossRef]
6. Potts, C.N.; Kovalyov, M.Y. Scheduling with batching: A review. *Eur. J. Oper. Res.* **2000**, *120*, 228–249. [CrossRef]
7. Hall, N.; Potts, C. The coordination of scheduling and batch deliveries. *Ann. Oper. Res.* **2005**, *135*, 41–64. [CrossRef]
8. Chen, Z.-L. Integrated production and outbound distribution scheduling: Review and extensions. *Oper. Res.* **2010**, *58*, 130–148. [CrossRef]
9. Wang, D.Y.; Grunder, O.; Moudni, A.E. Integrated scheduling of production and distribution operations: A review. *Int. J. Ind. Syst. Eng.* **2015**, *19*, 94–122. [CrossRef]
10. Zhong, X.; Jiang, D. Integrated Scheduling of Production and Distribution with Release Dates and Capacitated Deliveries. *Math. Probl. Eng.* **2016**, 9315197. [CrossRef]
11. Cheng, T.C.E.; Gordon, V.S.; Kovalyov, M.Y. Single machine scheduling with batch deliveries. *Eur. J. Oper. Res.* **1996**, *94*, 277–283. [CrossRef]
12. Yang X. Scheduling with generalized batch delivery dates and earliness penalties. *IEEE Trans.* **2000**, *32*, 735–742. [CrossRef]
13. Lee, C.Y.; Chen, Z.L. Machine scheduling with transportation considerations. *J. Sched.* **2001**, *4*, 3–24. [CrossRef]
14. Lu, L.; Zhang, L. A PTAS for single-machine scheduling with release dates and job delivery to minimize makespan. *RAIRO Oper. Res.* **2019**, *53*, 1261–1266. [CrossRef]
15. Averback, I.; Xue, Z. On-line supply chain scheduling problems with preemption. *Eur. J. Oper. Res.* **2007**, *181*, 500–504. [CrossRef]
16. Ng, C.T.; Lu, L. On-line integrated production and outbound distribution scheduling to minimize the maximum delivery completion time. *J. Sched.* **2012**, *15*, 391–398. [CrossRef]
17. Zhong, W.; Chen, Z.L. Flowshop scheduling with interstage job transportation. *J. Sched.* **2015**, *18*, 411–422. [CrossRef]
18. Condotta, A.; Knust, S.; Shakhlevich, N.V. Parallel batch scheduling of equal-length jobs with release and due dates. *J. Sched.* **2010**, *13*, 463–477. [CrossRef]
19. Li, W.; Yuan, J. Improved online algorithms for the batch scheduling of equal-length jobs with incompatible families to maximize the weighted number of early jobs. *Optim. Lett.* **2014**, *8*, 1691–1706. [CrossRef]
20. Li, S. Approximation algorithms for scheduling jobs with release times and arbitrary sizes on batch machines with non-identical capacities. *Eur. J. Oper. Res.* **2017**, *263*, 815–826. [CrossRef]
21. Xu, S.; Bean, J.C. Scheduling parallel-machine batch operations to maximize on-time delivery performance. *J. Sched.* **2016**, *19*, 583–600. [CrossRef]

22. Fan, W.; Pei, J.; Liu, X.; Pardalos, P.M.; Kong, M. Serial-batching group scheduling with release times and the combined effects of deterioration and truncated order-dependent learning. *J. Glob. Optim.* **2018**, *71*, 147–163. [CrossRef]
23. Hsu, H.-P.; Wang, C.-N. Resources Planning for Container Terminal in a Maritime Supply Chain Using Multiple Particle Swarms Optimization (MPSO). *Mathematics* **2020**, *8*, 764. [CrossRef]
24. Ioan, B.; Mozi, R.M.; Lucian, G.; Gheorghe, F.; Horia, T.; Ioan, B.; Mircea-Iosif, R. An Empirical Investigation on Determinants of Sustainable Economic Growth. Lessons from Central and Eastern European Countries. *J. Risk Financ. Manag.* **2020**, *13*, 146. [CrossRef]
25. Vakhania, N.; Werner, F. A Brief Look at Multi-Criteria Problems: Multi-Threshold Optimization versus Pareto-Optimization. In *Multi-criteria Optimization—Pareto-Optimal and Related Principles*; Nodari, V., Frank, W., Eds.; InTech Open: London, UK, 2020. [CrossRef]