EURO Journal on Computational Optimization 12 (2024) 100095

Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

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

New computational results for integrated production and outbound distribution scheduling problems for a product with a short lifespan

Markó Horváth

HUN-REN Institute for Computer Science and Control, Kende u. 13-17., Budapest, H-1111, Hungary

ARTICLE INFO

Dataset link: https://github.com/hmarko89/PTSP-supplementary

Keywords:
Supply chain coordination
Scheduling
Transportation
Integration
Vehicle routing problem
Variable neighborhood search

ABSTRACT

In this paper, we consider an integrated production and outbound distribution scheduling problem with a single production site, and its extension to multiple plants. A set of orders must be satisfied such that the required pieces from a single product must be first processed on a single machine in a plant, then must be delivered to the customers before their lifespan expire using a single vehicle. The goal is to minimize the makespan of the solution, which is the return time of the vehicle after its last trip. We propose an elementary variable neighborhood search to solve the problem, using two new local search operators. Our computational results show that this simple procedure outperforms the existing, sometimes complex approaches on the widely used benchmark dataset. We also review the existing computational results, and demonstrate that in some cases the comparisons in the literature are invalid due to the use of different rounding of the data. By re-evaluating the accessible solutions we provide a fair comparison for each rounding method. We also consider the extension of the problem to multiple plants, and adapt our solution approach for this extension. Our experiments show that our method is competitive in terms of solution quality with the existing solution approach for the problem.

1. Introduction

Production and distribution are two interrelated functions of supply chains, which can be treated as independent problems, yet they have a huge impact on each other. These problems are often solved separately in a sequential way, which leads to suboptimal overall solutions as the problem solved for the first ignores the constraints and requirements of the other. The integration of these stages can help to obtain better overall solutions, but it also makes the problem more complex. Moreover, the problems can be examined in different details. For example, scheduling decisions for production (scheduling) and routing decisions for distribution (transportation) can be taken into account. In addition, production and distribution are often linked by intermediate inventory stages (storage), which brings another aspects and challenges to the proper coordination. There are several problems which address the integration of some processes. For example, *lot sizing problems* combine production and inventory management, the *inventory routing problems* integrate inventory management with transportation, and the *production routing problems* jointly optimize production, inventory management and transportation decisions.

Perishable (or time-sensitive) products (e.g., fresh food, nuclear medicine, daily newspaper) possibly cannot be stored due to their short shelf life, but must be delivered soon after production. In this case, inventory aspects may omitted, and the focus is on the integration of production and distribution.

E-mail address: marko.horvath@sztaki.hu.

https://doi.org/10.1016/j.ejco.2024.100095

Available online 20 June 2024

An *integrated production and outbound distribution scheduling problem* typically involves both machine scheduling and vehicle routing. Generally, some order requests (jobs) arrive at the beginning of the time horizon, which need to be processed on one or multiple machines, and then must be delivered to the customers using a single vehicle or a fleet of vehicles.

In this paper, we investigate integrated production and outbound distribution scheduling problems for a product with a short lifespan. We focus on the single-plant problem introduced by Geismar et al. [16], called the *Integrated Production and Transportation Scheduling Problem (PTSP)*, and its extension to multiple plants by Can Atasagun and Karaoğlan [9], called the *Integrated Production and Outbound Distribution Scheduling Problem with Multiple Plants and Perishable Items (IPODS-MP)*. Briefly stated, in the single-plant case, there is a set of customers, each of them has an order from the same product, which has a limited lifespan. The orders are first processed in the plant on a single machine, then delivered to the customers with a single vehicle with limited transportation capacity. Both the production and the transportation of the orders are done in batches. When the production of a batch is completed on the machine, the lifespan of its orders starts, and then the orders must be delivered in the same trip before they expire. The objective is to minimize the total time required to produce and deliver orders to meet all customer demands along with the return of the vehicle to the plant. In case of multiple plants, each order must be assigned to a plant, and each plant has its own vehicle to transport the orders.

Main contributions The purpose of the paper is to present new computational results for the PTSP, and also for the IPODS-MP. The contribution to the PTSP is twofold. On the one hand, we claim that the existing comparisons of the solution approaches for the PTSP in the literature are invalid due to some small yet significant differences in the data. By re-evaluating the accessible solutions, we aim to provide a more fair comparative study between the existing solution approaches. On the other hand, we propose a conceptually simple yet effective solution approach for the problem. Finally, we adapt this solution approach to the multiple-plant case, and show that our algorithm is also competitive in terms of solution quality with the existing approach for the IPODS-MP.

While studying the existing results from the literature for the PTSP, we discovered a small, but significant difference in the computational experiments, which (as we claim) makes the previous comparisons invalid. That is, although the authors used a common benchmark dataset, some of them rounded travel times differently as others. This seemingly small difference is aggravated to such an extent that it makes the previous results incomparable. In this paper, we demonstrate that the comparisons made between the previous results are invalid, and by re-evaluating the accessible solutions, we also provide a more fair comparative study between the existing results.

We propose an elementary local search-based approach for the PTSP. Two problem-specific local search operators are introduced, which are then used in a variable neighborhood search. Our computational experiments show that this conceptually simple procedure outperforms the existing, sometimes complex solution approaches on the widely used benchmark dataset. That is, we obtained better results for almost half of the benchmark instances than the existing methods for the problem (and our algorithm did not find the best known solution so far for only one instance). We also provide new lower bounds for the benchmark instances.

Recently, Can Atasagun and Karaoğlan [9] extended the PTSP to multiple plants. We adapt our solution approach to this extension, and our computational experiments show that this adaption is competitive in terms of solution quality with the existing solution approach for the problem, as our method found a better solution than the former approach for almost half of the instances.

Structure of the paper In Section 2, we position the investigated problems, and review the related literature. In Section 3, we define the PTSP in detail. We also investigate the structure of the solutions, and using our observations, we propose an evaluation procedure to check the feasibility and to calculate optimal makespan value for solutions given in a compact form. In Section 4, we present our local search-based approach. In Section 5, we introduce two lower bounds for the problem. In Section 6, we re-evaluate and assess the previous results from the literature. In Section 7, we evaluate our local search approach and compare the results to the existing ones. In Section 8, we investigate the extension of the problem to multiple plants (IPODS-MP), evaluate the adaption of our solution approach on this problem, and compare the results to the existing ones. Finally, we conclude the paper in Section 9.

2. Integrated production and outbound distribution scheduling

Recall that the *integrated production and outbound distribution scheduling (IPODS) problem* integrates machine scheduling and vehicle routing. In the basic version of the problem, some order requests (jobs) arrive at the beginning of the time horizon, which need to be processed on one or multiple machines, and then must be delivered to the customers using a single vehicle or a fleet of vehicles. That is, IPODSs integrate production and distribution stages with scheduling and routing details, ignoring inventory aspects.

In this section, we first position integrated production and outbound distribution scheduling problems among similar integrated problems (Section 2.1), then we position the investigated problems among IPODSs (Section 2.2), finally we review the literature related to the PTSP and the IPODS-MP (Section 2.3).

2.1. Relation to other integrated problems

There are several problems that address a combination of production, storage, and distribution stages. For example, *lot sizing problems* combine production with inventory management on the production side [18], *inventory routing problems* integrate vehicle routing with inventory management on the customer side [4,7,12,34], and *production (inventory) routing problems* jointly optimize production, inventory management and transportation decisions [1]. Unlike these problems, IPODSs do not consider storage aspects. In addition, these problems cover several periods, however, IPODSs are typically single-period problems.

Table 1
Integrated production and outbound distribution scheduling problems with lifespan constraints.

Article	Production		Distribution		Benchmark	
	sites	machines	vehicles	trips	single	multi
Geismar et al. [16]	1	1	1	M	⋆	
Karaoğlan and Kesen [21]	1	1	1	M	⋆	
Devapriya [13]	1, M	1	Ho$^{\times}$	M		
Devapriya et al. [14]	1	1	Ho	M		
Lacomme et al. [24,25]	1	1	Ho	M	⋆	
Can Atasagun and Karaoğlan [8]	M	1	Ho$^{\times}$	M		
Can Atasagun and Karaoğlan [9]	M	1	1$^{\times}$	M	⋆	⋆
this paper	1, M	1	1$^{\times}$	M	⋆	⋆
Armstrong et al. [3]	1	1	1	1		
Viergutz [35]	1	1	1	1		
Viergutz and Knust [36]	1	1	1	1		
Amorim et al. [2]	1	P	Ho	1		
Belo-Filho et al. [5]	1	P	Ho	1		
Kergosien et al. [22]	1	P	1	M		

$^{\times}$for each production site.

Other problems combine production and distribution, but unlike IPODSs, do not take scheduling or routing details into account. For example, some papers investigated machine scheduling problems, where the completed jobs must be assigned to vehicles with fixed departure times, however, the determining of vehicles routes is not part of the task [27,28].

2.2. Classification of IPODSs

There are several review papers that classify IPODSs in different ways [11,31,37,32,6]. For example, Chen [11] introduced a five-field notation for the problems by extending the well-known three-field notation of Graham et al. [19] for machine scheduling problems with two fields that indicate the characteristic of the delivery process and the number of customers. Berghman et al. [6] classified production environment into two categories. The first category contains those problems that require only one operation. These are, single machine problems at a single production site (*plant*), single machine problems at multiple production sites, and parallel machine problems. The second category contains those problems that require multiple operations (e.g., flow shop, open shop, job shop). The authors also classified the delivery methods, and divided problems into four categories based on whether one or multiple vehicles are available, and whether the vehicles can perform only one (*single-trip vehicles*) or multiple trips (*multi-trip vehicles*). According to the classification of Berghman et al. [6], the PTSP is a single machine problem at a single production site with one multi-trip vehicle, and the IPODS-MP is a single machine problem at multiple production sites with one multi-trip vehicle for each site.

2.3. IPODSs with lifespan constraints

Papers on IPODSs for perishable products handle the lifetime of the items in different ways. Although, some works were based on perishable products, lifetime was not taken into account at all [26,30,29]. Some papers ensured delivery before expiry with close time-windows [33]. Some researches considered quality decay, that is, the quality of the produced perishable items decreases according to a given function [10,15,20]. In case of other problems, such as the PTSP [16], a *lifespan constraint* is considered, that is, the maximum delay between the production and the delivery of an item or a batch is explicitly limited.

In the rest, we review those researches that consider IPODSs with lifespan constraints. We collect these papers in Table 1, where we indicate the basic characteristics of the investigated problems. These are, the number of production sites (*Production/sites*), the machine environment (*Production/machines*, 1: single machine, P: parallel machines), the type of the vehicle fleet (*Distribution/vehicles*, 1: single vehicle, Ho: homogeneous fleet), and the number of trips that a vehicle can perform (*Distribution/trips*, 1: single trip, M: multiple trips). We also indicate whether the benchmark datasets of Geismar et al. [16] and Can Atasagun and Karaoğlan [9] were used for computational experiments (*Benchmark/single* and *Benchmark/multi*, respectively). The first two rows refer to those papers, which dealt with exactly the PTSP. The rows until the next line refer to the extensions of the PTSP, including the IPODS-MP. Papers in the last rows consider problems which differ from the PTSP in multiple basic characteristics.

2.3.1. The PTSP

The PTSP was introduced by Geismar et al. [16]. The problem was motivated by a practical scheduling problem encountered at chemical manufacturer, where one of the products, has a short lifespan. The authors proved that the problem is NP-hard in the strong sense, and introduced several lower bounds on the optimum. They also applied a genetic algorithm and a memetic algorithm for the problem. The backbone of these two evolutionary algorithms is that for a given sequence of the orders a set of feasible batches is determined and improved with a *2-opt* heuristic, then the batches are sequenced by the algorithm of Gilmore and Gomory [17] resulting an optimal no-wait schedule. This schedule is compressed then, if possible, that is, production and transportation are

scheduled as early as possible. In addition, the authors introduced a benchmark dataset consisting of 72 instances and performed computational experiments to evaluate their solution approach. These instances were generated considering 40-50 customers (orders), three different space distributions for the customer locations, two different capacity values for the vehicle, two different lifespan values for the product, and three different production rates for the machine, see Section 6.1.1 for details.

Karaoğlan and Kesen [21] proposed a branch-and-cut procedure for the PTSP. The authors formulated the problem as a mixed-integer linear program and adapted two classes of valid inequalities from the vehicle routing problem literature, which were heuristically separated during the global search. They also presented a simulated annealing approach including local search operators *merge*, *swap*, *move*, and *2-opt*. This procedure was applied before the branch-and-cut to find an initial solution (and thus an upper bound) for the problem, and inside a rounding heuristic. The authors tested their solution approach on the benchmark dataset of Geismar et al. [16]. They evaluated the effects of the proposed valid inequalities, and compared their results to the former ones. Based on these experiments, branch-and-cut yielded significantly better results on instances with higher capacities than the heuristic algorithms of Geismar et al. [16], but it was not so effective on instances with lower capacities.

2.3.2. Extensions of the PTSP

Lacomme et al. [24] extended the PTSP to multiple vehicles (PTSPm). The authors proposed a greedy randomized adaptive search procedure combined with an evolutionary local search procedure for the problem. The backbone of this procedure is similar to the one of Geismar et al. [16], however, the key points differ from each other. That is, giant trips are splitted to smaller ones, which are first improved with a local search procedure including *2-opt*, *swap* and other job shop scheduling operators, then the resulted batches are scheduled using a disjunctive graph model. In their computational experiments, the authors compared their method on the single vehicle case to the one of Geismar et al. [16]. They also performed experiments on this dataset with multiple (2 and 3) vehicles. In [25], the authors extended their previous work. They also compared their approach on the single vehicle case to the one of Karaoğlan and Kesen [21], and extended their computational experiments up to 6 vehicles on a newly created dataset. Based on these results, the authors claimed that their solution approach is more effective on this dataset than the other approaches. All their results (including the found solutions) are available online at [23]. In the rest of this paper, these entire work of the authors is referred as [25].

Devapriya [13] and Devapriya et al. [14] considered an extension of the PTSP to multiple vehicles, with the aim to determine an optimal fleet size. In their problem, the number of vehicles is a decision variable, the planning horizon is limited, and the objective is to minimize a combination of travel costs and the fixed costs for hiring vehicles. The authors proposed a mixed-integer linear programming formulation for the problem, which, based on their experiments, could be solved to optimality in a reasonable time only for 4 customers. The authors also provided a genetic algorithm and two memetic algorithms for the problem. Devapriya [13] also considered the problem with multiple sites, where each plant has its own vehicle fleet.

Can Atasagun and Karaoğlan [8] extended the PTSP to multiple sites and multiple vehicles (PTSP-MP-MV). In their problem, there is a single machine and a fleet of homogeneous multi-trip vehicles at each plant. The objective is to minimize a combination of production, distribution, and vehicle costs. The authors provided a mixed-integer linear programming formulation for the problem, which were strengthened with several valid inequalities. Can Atasagun and Karaoğlan [9] considered a similar extension to multiple sites, however, in their problem only one vehicle is available at each plant (IPODS-MP). Similarly to the original problem, the objective is to minimize the makespan of the solution. The authors proposed a variable neighborhood search for the problem with local search operators *merge*, *route move*, *insert*, and *swap*. The authors presented computational results for the single-plant case on the benchmark dataset of Geismar et al. [16], and they also introduced a new benchmark dataset for the multi-plant problem. The new instances were generated considering 10-100 customers (orders), 2-5 plants, three different space distributions for the customer locations, two different demand distributions for the orders, two different capacity values for the vehicles, two different lifespan values for the product, and three different production rates for the machines, see Section 8.2.1 for details.

2.3.3. Problems related to the PTSP

Some papers dealing with IPODSs with lifespan constraints, however, the considered problems fundamentally differ from the PTSP and the IPODS-MP.

Armstrong et al. [3] considered a problem with a single machine and a single vehicle, which can perform only a single trip. In this problem, it is not necessary to visit all customers, but the delivery sequence of the visited customers, which is the same as the production sequence, must follow a predetermined order. Each customer has a time window within which their delivery is feasible. The orders have a uniform lifetime, i.e., an upper bound for the time elapsed between the end of the production and the delivery. The task is to choose a subset of customers, such that their demands can be fulfilled in the given visiting order, and the total amount of demand delivered is maximized. The authors proposed a branch-and-bound procedure to solve the problem. Viergutz [35] and Viergutz and Knust [36] proposed a corrected variant of the branch-and-bound procedure of Armstrong et al. [3], where a mistake in the upper bound calculation was fixed. The authors also considered a variant of the problem, where the production and delivery sequences might differ and are not fixed in advance.

Amorim et al. [2] considered a problem with parallel machines, a homogeneous fleet of single-trip vehicles, and multiple products. Perishable products have a given shelf life limiting the maximal elapsed time between the start of the production and the delivery. The objective is to minimize a combination of setup, productions, and transportation costs. The authors proposed two mixed-integer linear programming formulations for the problem. The first formulation includes scheduling decisions, while the second formulation integrates lot sizing and vehicle routing. The authors evaluated their formulations with a commercial MIP solver. Belo-Filho et al. [5] addressed the same problem as in [2]. The authors proposed an adaptive large neighborhood search framework to tackle the problem.

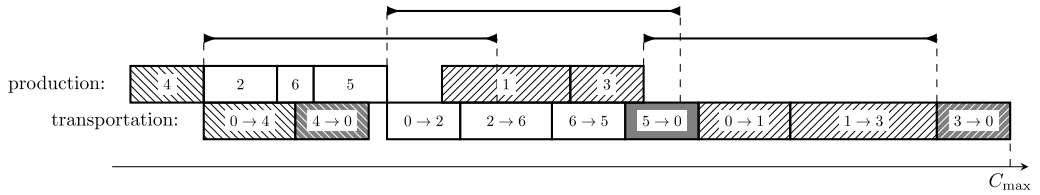

Fig. 1. Example for a solution. Six orders are grouped into three batches depicted with different patterns. Lifespan intervals are depicted above the GANTT chart.

Kergosien et al. [22] considered a problem with uniform parallel machines and a multi-trip vehicle. Some jobs have a *stability time*, which is an upper limit for the time elapsed between the start of the production and the delivery. The objective is to minimize the maximum tardiness of delivery. The authors provided a mixed-integer linear programming formulation for the problem, and proposed a solution approach based on Benders decomposition.

3. Problem statement

In this section, we define the PTSP in detail. In Section 3.1, we first provide an overview of the problem, while the formal definition is given in Section 3.2. In Section 3.3, we investigate the structure of solutions, and based on the observations, we present an evaluation procedure to calculate the optimal makespan value for a fixed batch-sequence. This procedure will be included in our local search method (see Section 4.3), and will be also used to check and to re-evaluate the accessible solutions for the benchmark dataset (see Section 6.2).

3.1. Problem overview

There is a set of customers, each of them has an order from the same product, which has a limited lifespan. The orders are first processed in a plant on a single machine, then delivered to the customers with a single vehicle with limited transportation capacity. Both the production and the transportation of the orders are done in batches. When the production of a batch is completed on the machine, the lifespan of its orders starts, and all orders must be delivered in the same trip before they expire. The objective is to minimize the total time required to produce and deliver orders to meet all customer demands along with the return of the vehicle to the plant. In other words, the goal is to minimize the return time of the vehicle after its last trip.

Similarly to Geismar et al. [16], Karaoğlan and Kesen [21], Lacomme et al. [25], and Can Atasagun and Karaoğlan [9], we assume that production and transportation follow a first-produced-first-transported rule. By this, the problem with fixed batches can be reduced to a two-machine permutation flow shop problem with maximum delays. That is, the batches are considered as jobs, the first machine refers to production (processing time of a job is equal to the total production time of the corresponding batch), the second machine refers to transportation (processing time of a job is equal to the total travel time of the corresponding trip), and maximum delays can be derived from the lifespan value. The goal is to minimize the *makespan* of the schedule, which is the completion time of the last job on the second machine.

Example In Fig. 1, we depict a solution for an instance of the PTSP, which is based on the above analogy. Six orders are grouped into three batches, which are indicated with different patterns (north-west lines, blank fill, north-east lines). The last elements of the batches on the transportation side (depicted with gray background) refer to the vehicle's returns to the plant.

3.2. Formal definition

The index set of the customers and the orders are the same, and denoted with $\mathcal{O} = \{1, \dots, n\}$. The demand of customer i (i.e., the size of order i) is q_i.

Batches The orders must be divided into non-empty batches, say $\mathcal{B}_1, \dots, \mathcal{B}_m$ ($1 \le m \le n$ is not a pre-fixed constant), where each batch $\mathcal{B}_j \subseteq \mathcal{O}$ is an ordered subset of the orders. A batch must be first processed on the machine, then its orders must be delivered to the customers in a single trip.

Production The machine at the plant has a production rate $r > 0$, and the orders of a batch are processed one after another without any idle times between them and without preemption. That is, the *production time* of a batch $\mathcal{B}_j$ is

$$p_j = \frac{1}{r} \cdot \sum_{i \in \mathcal{B}_j} q_i,$$

i.e., if the production of the batch starts at time point t, then it is completed at $t + p_j$. Right after an entire batch is completed, the lifespan B of its orders starts, that is, if a batch is completed at time point t, then all of its orders must be delivered until $t + B$. Notice that the production order in a batch is irrelevant, since reordering does not change the production time, and the common expiration time starts only after the last order is completed. Thus, it is assumed without loss of generality that the production sequence is the same as the transportation sequence.

Transportation A vehicle with capacity of Q is used to transport orders from the plant to the customers ($\max_i q_i \le Q$). At the beginning, the vehicle is empty, and it is located at the plant. The travel time between customers i and j (where 0 refers to the plant) is τ_{ij}. These times are symmetric (i.e., $\tau_{ij} = \tau_{ji}$ for all i, j), satisfy the triangle inequality (i.e., $\tau_{ij} + \tau_{jk} \ge \tau_{ik}$ for all i, j, k), and $\tau_{0,j} \le B$ holds for all j. Loading and unloading times are negligible. Then, the *transportation time* of the batch $\mathcal{B}_j = (i_1, \dots, i_\ell)$ is

$$t_j = \tau_{0,i_1} + \sum_{k=1}^{\ell-1} \tau_{i_k,i_{k+1}} + \tau_{i_\ell,0},$$

where ℓ, the index of the last order, depends on the batch.

Solution The solution for the problem is a *batch-sequence*, $(\mathcal{B}_1, \dots, \mathcal{B}_m)$, along with their completion times on the production side, $C_1^p \le \dots \le C_m^p$, and on the transportation side, $C_1^t \le \dots \le C_m^t$. Notice that $C_j^p - p_j$ and $C_j^t - t_j$ refer to the start times of batch $\mathcal{B}_j$. A solution is *feasible* if the following constraints, (1)–(6), are satisfied. The batches must partition the orders, i.e.,

$$\bigcup_{j=1}^{m} \mathcal{B}_j = \mathcal{O} \quad \text{with} \quad \mathcal{B}_j \cap \mathcal{B}_k = \emptyset \text{ for all } j \ne k. \tag{1}$$

Batches cannot exceed the vehicle's capacity limit, i.e.,

$$\sum_{i \in \mathcal{B}_j} q_i \le Q \quad \text{for all } j. \tag{2}$$

Completion times must respect the scheduling constraints, that is, start times are non-negative, and batches cannot overlap either on the production side or the transportation side, i.e.,

$$p_1 \le C_1^p \text{ and } C_{j-1}^p + p_j \le C_j^p \quad \text{for all } 1 < j, \tag{3}$$

$$t_1 \le C_1^t \text{ and } C_{j-1}^t + t_j \le C_j^t \quad \text{for all } 1 < j, \tag{4}$$

and transportation cannot start before production ends, i.e.,

$$C_j^p + t_j \le C_j^t \quad \text{for all } j. \tag{5}$$

The solution satisfies the lifespan constraint, if

$$C_j^t - \tau_{i_\ell,0} \le C_j^p + B \quad \text{for all } j. \tag{6}$$

Note that constraints (5) and (6) result that

$$t_j - \tau_{i_\ell,0} \le B \quad \text{for all } j, \tag{7}$$

that is, the total travel time until the last customer in a trip cannot exceed the lifespan.

Objective The objective of the problem is to minimize the *makespan* $C_{\max}$ of the solution, where $C_{\max} = \max_{j=1,\dots,m} C_j^t = C_m^t$ is the completion time of the transportation of the last batch.

3.3. Notes on the structure of the solutions

In the following, we investigate the structure of the feasible solutions (Section 3.3.1). Based on our observations, we propose an evaluation procedure to calculate the optimal makespan value for a fixed batch-sequence (Section 3.3.2).

3.3.1. Compressed solutions

Consider a feasible solution with batch-sequence $(\mathcal{B}_1, \dots, \mathcal{B}_m)$, and completion times $C_1^p, \dots, C_m^p$ and $C_1^t, \dots, C_m^t$. We say that the solution is *compressed*, if by keeping the given batch-sequence, all production and transportation are completed as early as possible. Equivalently, all production and transportation start as early as possible. Formally, there is no other feasible solution with exactly the same batch-sequence, and completions times $\tilde{C}_1^p, \dots, \tilde{C}_m^p$ and $\tilde{C}_1^t, \dots, \tilde{C}_m^t$, such that $\tilde{C}_j^p < C_j^p$ or $\tilde{C}_j^t < C_j^t$ holds for some batch $\mathcal{B}_j$.

It follows from the definition that a compressed solution, if any, is unique for the given batch-sequence, since the presence of two distinct compressed solutions would contradict the definition. Moreover, the compressed solution, if exists, is always optimal for the given batch-sequence, since there is no other feasible solution with $\tilde{C}_m^t < C_m^t$.

Now, consider the following procedure, which extends a given batch-sequence $(\mathcal{B}_1, \dots, \mathcal{B}_m)$ with completion times to obtain a feasible solution. Clearly, constraints (1), (2) and (7) for the batches must be met in advance. Production and transportation of the batches are scheduled in the given sequence, such that each batch is scheduled as early as possible, considering the feasibility constraints. Production of the first batch starts at 0, and its transportation starts right after production ends (i.e., at $C_1^p = p_1$). In an intermediate state of the procedure, production and transportation of a batch $\mathcal{B}_j$ are scheduled to start right after the previous operations end (i.e., at C_{j-1}^p and C_{j-1}^t, respectively). If transportation begins before production ends (i.e., $C_{j-1}^t < C_j^p$), then the start

of transportation is postponed to the end of production (i.e., to C^p_j) to satisfy constraint (5). Otherwise, if production ends too early compared to transportation, that is, the lifespan constraint is violated, then the start of production is postponed to the earliest time with which the constraint (6) is satisfied. For details, see Section 3.3.2.

To sum up our observations, for each fixed batch-sequence there exists a unique compressed solution, which, moreover, is optimal for the problem among the solutions with the same batch-sequence. Thus, it is sufficient to search for an optimal solution among compressed solutions. In addition, a compressed solution can be fully described with only the corresponding batch-sequence, since the appropriate completion times can be determined by, for example, the sketched procedure.

Example The solution depicted in Fig. 1 is compressed, since none of the batches can be moved earlier. The transportation of the first batch starts as soon as its production is finished. The production of the second batch starts right after the first one is completed on the machine. The transportation of the second batch cannot start earlier, since the production just finishes. The production of the third batch cannot start right after the second one is finished, since the lifespan constraint would be violated due to the vehicle not being available for too long. Thus, the start of this production is postponed to the earliest time that can meet the expiration deadline, and the last order is delivered at the last possible moment. The solution is represented by batch-sequence $((4),(2,6,5),(1,3))$.

3.3.2. Evaluation procedure

In the following, we describe an evaluation procedure which returns the optimal makespan value for a given batch-sequence. Note that Geismar et al. [16] also proposed a method for the same purpose, which is based on a linear programming formulation reduced to a shortest path problem. Our procedure also checks whether the given batches fulfill the capacity constraint (2) and the lifespan constraint (7).

Algorithm 1 Procedure `evaluate`: checks the feasibility of the given solution (batch-sequence), and calculates its optimal makespan value.

Input $\mathcal{B}=(\mathcal{B}_1,\dots,\mathcal{B}_m)$: batch-sequence, Q: capacity, B: lifespan, r: production rate, τ: travel times, *checkFeasibility*: boolean to check feasibility

1: $C^p_j \leftarrow 0$ for all $j=0,\dots,m$
2: $C^t_j \leftarrow 0$ for all $j=0,\dots,m$
3: $\xi \leftarrow$ false ▷ *infeasible?*
4: **for** $\mathcal{B}_j=(i_1,\dots,i_\ell)\in\mathcal{B}$ **do**
5: $\quad p \leftarrow \frac{1}{r}\cdot\sum_{k=1}^{\ell} q_{j_k}$ ▷ *total prod. time*
6: $\quad$ **if** $Q < r\cdot p$ **then** ▷ *check capacity constraint*
7: $\quad\quad \xi \leftarrow$ true
8: $\quad t \leftarrow \tau_{0,i_1}+\sum_{k=1}^{\ell-1}\tau_{i_k,i_{k+1}}$ ▷ *transp. time until last customer*
9: $\quad \Delta \leftarrow B-t$ ▷ *max delay*
10: $\quad$ **if** $\Delta<0$ **then** ▷ *check lifespan constraint*
11: $\quad\quad \xi \leftarrow$ true
12: $\quad\quad \Delta \leftarrow 0$
13: $\quad t \leftarrow t+\tau_{i_\ell,0}$ ▷ *total transp. time*
14: $\quad \bar{p} \leftarrow C^p_{j-1}+p$ ▷ *earliest prod. finish*
15: $\quad$ **if** $C^t_{j-1}\le\bar{p}$ **then** ▷ *vehicle will be available at earliest prod. finish*
16: $\quad\quad C^p_j \leftarrow \bar{p}$
17: $\quad\quad C^t_j \leftarrow C^p_j+t$
18: $\quad$ **else** ▷ *vehicle will not be available at earliest prod. finish*
19: $\quad\quad C^p_j \leftarrow \max\{\bar{p}, C^t_{j-1}-\Delta\}$
20: $\quad\quad C^t_j \leftarrow C^t_{j-1}+t$
21: **if** ξ and *checkFeasibility* **then**
22: $\quad$ **return** ∞
23: **return** C^t_m ▷ *makespan*

Procedure `evaluate`, see Algorithm 1, requires the solution as a batch-sequence, the capacity value Q, the lifespan value B, the production rate r, and the travel times τ between locations. The procedure also gets the parameter *checkFeasibility* indicating whether the feasibility of the solution must be checked.

Completion times C^p_j and C^t_j are initially set to zero for all $j=0,\dots,m$, and will be calculated during the procedure. Variable ξ indicates whether the given solution is infeasible. Going through the batches in sequence, we calculate their total production time p, their total transportation time t, and the maximum delay Δ between the production finish and the transportation start (lines 5-13). Then, we schedule production and transportation of the incumbent batch as early as possible. Actually, we schedule production right after the previous production is finished, and we schedule transportation right after the previous transportation is finished. If transportation starts earlier than production is finished, we adjust the transportation start time (lines 15-17). For an example, see the first two batches in Fig. 1. Otherwise, if the maximum delay between production and transportation is violated, then the production start time is adjusted (lines 18-20). For and example, see the third batch in Fig. 1. Finally, the completion time of the last transportation (i.e., the makespan value of the solution) is returned (line 23).

The feasibility of the solution is checked in two places. For each batch, we first check if it exceeds the capacity limit (line 6), then we check if the total travel time until the last customer violates the lifespan constraint (line 10). Note that in case of batches violating lifespan constraint, Δ must be set to zero (line 12) to avoid overlap between production and transportation (see line 19). If

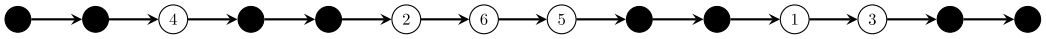

Fig. 2. The canonical representation of the solution depicted in Fig. 1. Batches (sequences of white circles) are separated with exactly two plant nodes (black circles).

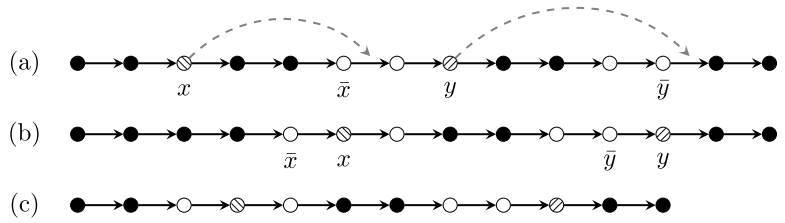

Fig. 3. Example for a knockout operation. (a) Initial solution. (b) Solution after change. (c) Solution after bringing it to canonical form again.

the solution is infeasible and verification is on (that is, *checkFeasibility* is true), then ∞ (i.e., an appropriately big number) is returned instead of the makespan value (line 22).

4. Variable neighborhood search for the PTSP

In this section, we present our local search-based solution approach to solve the PTSP. In Section 4.1, we propose our modeling approach including our solution representation. In Section 4.2, we introduce two problem-specific local search operators, which are then used in a variable neighborhood search, described in Section 4.3.

4.1. Modeling approach

In the rest of this section, we use a *route-based solution representation*. There are two types of nodes, namely, *plant nodes* and *customer nodes*, where the former ones represent the plant, and the latter ones represent the customers. For each customer, there is only one customer node in the solution. Customer nodes of the same batch follow each other in their transportation (and production) order, and two consecutive batches are separated with one or more plant nodes.

A solution representation is called *canonical*, if it starts and ends with exactly two plant nodes, and consecutive batches are separated by exactly two plant nodes. In Fig. 2 we depict the canonical solution corresponding to solution in Fig. 1. Note that this representation is something like two non-empty batches are always separated by an empty batch. The advantage of such a representation is that during local search we can easily create new batches, or insert a complete batch between two other batches by inserting customers nodes between two plant nodes. Each solution can be brought to canonical form by adding and/or removing plant nodes. That is, lonely plant nodes must be doubled, and plant node sequences must be reduced to two-long ones.

Note that the corresponding batch-sequence can be easily obtained from this representation. Thus, the evaluation procedure described in Section 3.3.2 can be also used for the route-based solution representation to check whether a solution is feasible and to calculate its optimal makespan value.

4.2. Local search operators

In a local search approach one or more operators are used to move from solution to solution in the search space toward a local optimum. That is, an operator o describes possible local changes on an incumbent solution S, which defines a neighborhood $\mathcal{N}^o(S)$ consisting of those feasible solutions which can be resulted from S by applying a single change. For example, the *move operator* mentioned in the literature review relocates a single customer node, that is, removes it from its current position, and inserts to another place ($\mathcal{N}^{move}(S)$). The *swap operator* exchanges two customer nodes ($\mathcal{N}^{swap}(S)$). In the following, we define two problem-specific local search operators which were proved to be effective on this problem based on our preliminary experiments.

Knockout operator The basic idea of the *knockout operator* is to relocate an order to another batch, and, if needed and possible, throw something out of this batch to keep the solution feasible. This operator works on a node-tuple $(x, y, \bar{x}, \bar{y})$, where x and y are two customer nodes from distinct batches, $\bar{x} \neq x$ is either a customer node in the current batch of y or the plant node that precedes the batch of y, and $\bar{y} \neq y$ is a node in the solution. The operator (i) removes x from its current position, (ii) inserts x after $\bar{x}$, (iii) removes y from its current position, and (iv) inserts y after $\bar{y}$. An example is shown in Fig. 3.

Notice that if $\bar{y}$ is the current predecessor of y in the solution, then y is not actually relocated, thus $\mathcal{N}^{move}(S) \subseteq \mathcal{N}^{knockout}(S)$. Also note that if $\bar{x}$ is the current predecessor of y and $\bar{y}$ is the current predecessor of x, then x and y are actually swapped, thus $\mathcal{N}^{swap}(S) \subseteq \mathcal{N}^{knockout}(S)$.

Double relocation The basic idea of the *double relocation operator* is to create a new batch or extend an existing one with two orders. This operator works on a node-tuple $(x, y, \bar{x})$, where x and y are two distinct customer nodes, and $\bar{x} \notin \{x, y\}$ is a node in the solution. The operator (i) removes x from its current position, (ii) removes y from its current position, (iii) inserts x after $\bar{x}$, and (iv) inserts y after x. An example is shown in Fig. 4.

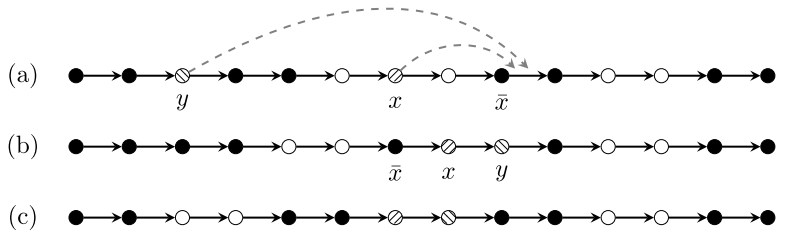

Fig. 4. Example for a double relocation. (a) Initial solution. (b) Solution after change. (c) Solution after bringing it to canonical form again.

4.3. Variable neighborhood search

In the following, we briefly present our local search-based approach for the PTSP. Each solution is associated with a makespan value calculated by the evaluation procedure described in Section 3.3.2. We say that a solution is better then an other one, if its evaluated value is less then that of the other. Our procedure consists of a given number of rounds. In each *round*, an initial solution is created, where the orders are inserted one after another into a separate batch. With the exception of the first round, we randomly shuffle this initial solution. Then, the knockout and the double relocation operators are used in a variable neighborhood search to improve the solution (i.e., to find a better one). We use the knockout operator as long as we can improve the incumbent solution, and double relocation operator is used only to get out of a local optimum with respect to the knockout operator. Finally, the best solution found in these rounds is resulted.

Step 1 (Initial solution) We start from the batch-sequence $((1),(2),\dots,(n))$, and if we are not in the first round, we randomly shuffle the batches. After that, we create the corresponding canonical route-based solution.

Step 2 (Knockouts) We repeat knockout operator for all possible node-tuples on the incumbent solution, and implement the best improving movement, if any. If the solution is improved, we bring it into canonical form, and we repeat Step 2.

Step 3 (Double relocations) We repeat double relocation operator for all possible node-tuples on the incumbent solution, and implement the best improving relocation, if any. If the solution is improved, we bring it into canonical form, and we go to Step 2.

Step 4 (Check round limit) If the round limit is reached, we terminate the procedure by returning the best solution found so far. Otherwise, we go to Step 1.

5. Lower bounds

In this section, we propose two valid lower bounds on the optimum of the PTSP. Both lower bounds exploit the machine scheduling representation of the problem, namely, the makespan value of a solution is at least the sum of the total processing time of the jobs on one machine and the minimum processing time on the other machine.

On the one hand, the total processing time on the first machine is independent of the batches (it is equal to the total production time of the orders), thus the first lower bound can be easily calculated as:

$$LB_1 = \frac{1}{r} \cdot \sum_{i=1}^{n} q_i + \min_{i=1,\dots,n} \{\tau_{0,i} + \tau_{i,0}\},$$

where the second term refers to the travel time of the shortest round trip, which is a valid lower bound on the length of the shortest trip due to the triangle inequality.

On the other hand, the total processing time on the second machine depends a lot on the determined batches, since it is equal to the total travel time of the corresponding trips. Thus, determining the smallest possible value is equivalent to solve a vehicle routing problem. For this, we propose the following straightforward integer programming formulation:

$$z_{\mathrm{IP}} = \min \left\{ \sum_{t \in \mathcal{T}} \tau(t) \cdot \mathbf{x}_t \ : \ \sum_{i \in t} \mathbf{x}_t \geq 1 \text{ for all } i = 1,\dots,n,\ \mathbf{x}_t \in \{0,1\} \text{ for all } t \in \mathcal{T} \right\},$$

where $\mathcal{T}$ is the set of all feasible trips, each trip t is associated with a binary variable $\mathbf{x}_t$ indicating whether the trip is performed or not, value $\tau(t)$ denotes the total travel time of trip t, and the constraints ensure that each customer is visited at least once. By this, the second lower bound is:

$$LB_2 = z_{\mathrm{IP}} + \frac{1}{r} \cdot \min_{i=1,\dots,n} q_i,$$

where the second term is the minimum production time among the orders. Then, the final lower bound is $LB = \max\{LB_1, LB_2\}$.

Note that the number of feasible trips may be too large to determine set $\mathcal{T}$ and/or to determine z_{IP}. Also note that the trips visiting the same subset of customers result the same columns in the problem matrix, thus, it is sufficient to keep the ones with the shortest total travel times.

Geismar et al. [16] introduced three lower bounds for the PTSP, also based on the above approach. The first one is the same as LB_1, however, the authors obtained the other two bounds by determining only lower bounds on z_{IP}. Thus, LB_2 is stronger (more precisely, not weaker) than their latter two bounds. Lacomme et al. [25] proposed an algorithm to calculate a Jackson rule-based lower bound for the PTSPm, which actually results the lower bound LB_1. Karaoğlan and Kesen [21] did not introduce lower bounds for the PTSP, however, their branch-and-cut procedure obviously results valid lower bound for each instance on which it is performed.

6. Revision of the previous results from the literature

Geismar et al. [16], Karaoğlan and Kesen [21], Lacomme et al. [25], and Can Atasagun and Karaoğlan [9] tested their solution approaches on the same benchmark dataset, which allowed the latters to make comparisons with previous results. We claim, however, that some of the comparisons are not valid, moreover, we will show some mistakes in the published results of Lacomme et al. [25].

In Section 6.1, we revise the existing results on the widely-used benchmark dataset, and raise our issues about the previous comparisons. In Section 6.2, we present the results of our comparative study, where we re-evaluated the accessible solutions to make a fair comparison between the existing results. In Section 6.3, we conclude the revision of the existing results.

Due to the length of the article, we do not discuss our concerns in detail, and do not provide detailed results of our experiments. However, we provide an online available supplementary material, where all the details can be found.

6.1. Previous results and comparisons

In the following, we introduce the benchmark dataset in question (Section 6.1.1), provide lower bounds for this dataset (Section 6.1.2), review existing results on this dataset (Section 6.1.3), and raise our concerns about previous comparisons (Section 6.1.4).

6.1.1. Benchmark dataset

The benchmark dataset introduced by Geismar et al. [16] is based on six *customer-instances*. Instances 1-3 consider 40 customers, and there are 50 customers in Instances 4-6. The locations of the customers are randomly generated from a two-dimensional uniform distribution of equal length and width. The plant is located in the center (that is, at $(0,0)$) of each of these squares, which have the size 200×200 (Instance 1 and 4), 300×300 (Instance 2 and 5), and 400×400 (Instance 3 and 6). Demands are randomly generated from interval $[100,300]$ using uniform distribution.

Each customer-instance is examined with 12 different *scenarios*, namely, with two capacity values ($Q\in\{300,600\}$), two lifespan values ($B\in\{300,600\}$), and three production rates ($r\in\{1,2,3\}$). These $6\times2\times2\times3=72$ different *instances* constitute the benchmark dataset.

The travel times between locations are based on their Euclidean distance, assuming that the vehicle speed is 1 unit of distance per unit of time. That is, the travel time between locations i and j with coordinates (x_i,y_i) and (x_j,y_j), respectively, is

$$\tau_{ij}=\sqrt{(x_j-x_i)^2+(y_j-y_i)^2}.$$

Note that these values are sometimes irrational. We will show that rounding differences caused the trouble in the previous comparisons. For later use, we also introduce notation

$$\check{\tau}_{ij}=\lfloor\tau_{ij}\rfloor \quad\text{and}\quad \ddot{\tau}_{ij}=\lfloor\tau_{ij}\times100\rfloor/100,$$

i.e., in case of *$\check{\tau}$-values*, the travel times rounded down to integer values, and in case of *$\ddot{\tau}$-values*, the travel times are rounded to two decimal places.

6.1.2. Lower bounds

We proposed two lower bounds for the PTSP in Section 5. In Table 2, we indicate the average lower bounds over the six customer-instances for each scenario (Q,B,r) with respect to $\check{\tau}$- and $\ddot{\tau}$-values. Recall that $LB=\max\{LB_1,LB_2\}$. Lower bounds LB_2 were determined by using `Python` package `Python-MIP` with the COIN-OR branch-and-cut solver (CBC).

Note that Geismar et al. [16] did not provide their calculated lower bounds for this dataset. They only provided the average values of their best found solutions and the average gaps for the scenarios, from which average lower bounds can be determined only within a certain tolerance. Similarly, Karaoğlan and Kesen [21] did not provide either exact lower bounds for these instances. Lacomme et al. [25] published their obtained lower bounds, which are similar to columns LB_1, see [24, Table 4] and [25, Table 7].

6.1.3. Previous results

Geismar et al. [16] summarized their computational results in a single table, where only the average values over the six customer-instances are indicated for the 12 scenarios. Moreover, these averages are rounded down to integer values. Since the authors did not provide the solutions themselves or their makespan value, we will publish further results in the same form. We copied the results of authors from [16, Table 7] to column $G/\check{C}_{\max}$ of Table 3 referring to that the authors used $\check{\tau}$-values in their experiments.

Table 2
Average lower bounds for each scenario with respect to $\check{\tau}$- and $\ddot{\tau}$-values.

Q	B	r	w.r.t. $\check{\tau}$-values			w.r.t. $\ddot{\tau}$-values		
			LB_1	LB_2	LB	LB_1	LB_2	LB
300	300	1	8 780.83	9 167.83	9 976.00	8 781.53	9 203.13	9 999.85
300	300	2	4 411.42	9 114.92	9 114.92	4 412.11	9 150.21	9 150.21
300	300	3	2 954.94	9 097.28	9 097.28	2 955.64	9 132.57	9 132.57
300	600	1	8 780.83	9 167.83	9 976.00	8 781.53	9 203.13	9 999.85
300	600	2	4 411.42	9 114.92	9 114.92	4 412.11	9 150.21	9 150.21
300	600	3	2 954.94	9 097.28	9 097.28	2 955.64	9 132.57	9 132.57
600	300	1	8 780.83	4 803.33	8 780.83	8 781.53	4 828.98	8 781.53
600	300	2	4 411.42	4 750.42	5 152.25	4 412.11	4 776.07	5 170.09
600	300	3	2 954.94	4 732.78	4 732.78	2 955.64	4 758.43	4 758.43
600	600	1	8 780.83	4 773.50	8 780.83	8 781.53	4 798.38	8 781.53
600	600	2	4 411.42	4 720.58	5 122.42	4 412.11	4 745.46	5 139.48
600	600	3	2 954.94	4 702.94	4 702.94	2 955.64	4 727.82	4 727.82

Table 3
Existing and re-evaluated results: average makespan values for the scenarios.

Q	B	r	G		KK		L		L-rev		CAK	
			$\ddot{C}_{\max}$	$\check{C}_{\max}$	$\ddot{C}_{\max}$	$\check{C}_{\max}$	$\ddot{C}_{\max}$	$\check{C}_{\max}$	$\ddot{C}_{\max}$	$\check{C}_{\max}$	$\ddot{C}_{\max}$	$\check{C}_{\max}$
300	300	1	-	10 040	10 049	-	-	10 039	10 062	10 039	10 086	-
300	300	2	-	9 157	9 179	-	-	9 148	9 183	9 148	9 233	-
300	300	3	-	9 125	9 153	-	-	9 118	9 154†	9 118	9 207	-
300	600	1	-	10 041	10 048	-	-	10 026	10 049	10 026	10 086	-
300	600	2	-	9 171	9 180	-	-	9 150	9 186	9 150	9 233	-
300	600	3	-	9 153	9 152	-	-	9 122	9 157	9 122	9 207	-
600	300	1	-	8 781	8 782	-	-	8 782	8 788†	8 786†	8 782	-
600	300	2	-	5 744	5 346	-	-	5 347	5 366†	5 348†	5 384	-
600	300	3	-	5 421	4 887	-	-	4 919	4 943†	4 918†	5 020	-
600	600	1	-	8 781	8 782	-	-	8 781	8 782	8 781	8 782	-
600	600	2	-	5 724	5 276	-	-	5 300	5 318	5 300	5 358	-
600	600	3	-	5 403	4 861	-	-	4 875	4 900	4 875	4 995	-
average				8 045	7 891			7 884	7 907	7 884	7 948	

[G] genetic algorithm of Geismar et al. [16].
[KK] branch-and-cut procedure of Karaoğlan and Kesen [21].
[L] greedy randomized adaptive search procedure of Lacomme et al. [25].
[L-rev] re-evaluated solutions of Lacomme et al. [25].
[CAK] variable neighborhood search of Can Atasagun and Karaoğlan [9].
† including infeasible solutions.

In addition to a similar summary table, Karaoğlan and Kesen [21] published the makespan values of their best solutions for all 72 instances, however, they could not provide us with the solutions themselves. Based on their results, and based on our experiments as well, we can state that the authors used $\ddot{\tau}$-values in their computational experiments, that is, they rounded travel times to two decimal places. The averages of their provided makespan values are indicated in column $KK/\ddot{C}_{\max}$ of Table 3 referring to that the authors used $\ddot{\tau}$-values. Note that this column completely matches the appropriate one in [21, Table 4] up to a rounding tolerance.

Lacomme et al. [25] repeated their solution procedure five times on each instance, and the authors made these $5 \times 72 = 360$ solutions available online, which allows us to check and to re-evaluate them with arbitrary travel time values by using the procedure described in Section 3.3.2. Their results and our experiments clearly prove that the authors used $\check{\tau}$-values in their computational experiments, that is, they rounded Euclidean distance based travel times down to integer values. The averages of their provided makespan values are indicated in column $L/\check{C}_{\max}$ of Table 3 referring to that the authors used $\check{\tau}$-values in their experiments. Recall that in this case each row refers to the average of not 6, but 30 solution values as the authors made 5 replications for each instance. Note that this column completely matches the appropriate ones in [24, Table 3] and [25, Table 7], which confirms that the authors indeed calculated the averages based on these provided makespan values.

Can Atasagun and Karaoğlan [9] did not provide detailed results for their experiments on the benchmark dataset, but published their summary in the same form as in [16]. Based on their detailed results for multi-plant instances, the authors used $\ddot{\tau}$-values for their tests, and thus we copied their average values from [9, Table 2] to column $CAK/\ddot{C}_{\max}$ of Table 3.

6.1.4. Our concerns about previous comparisons

We showed that Geismar et al. [16] and Lacomme et al. [25] used different travel time values for their computational experiments than Karaoğlan and Kesen [21] and Can Atasagun and Karaoğlan [9]. Although this does not seem like a big difference in itself, due to the characteristics of the instances it actually means that some authors compared the results of experiments performed on different datasets. Just to mention an example, Lacomme et al. [25] and Karaoğlan and Kesen [21] found the optimal makespan value of the very first instance, however, these optimal values differ using the two types of travel times, namely, these are 8 211.00 and 8 212.74, respectively, which (due to inadequate communication of differences) falsely suggests that the method of Lacomme et al. [25] found better solution on this instance than the one of Karaoğlan and Kesen [21]. It is also possible (and it actually happened) that a solution is feasible with respect to $\check{\tau}$-values, but infeasible w.r.t. $\ddot{\tau}$-values, which can be another criticism of the comparison, since the set of feasible solutions for an instance may differ if different travel times are used. As we will show in the next section, there are some other issues with the solutions provided by Lacomme et al. [25].

6.2. A fair comparative study (re-evaluation of the accessible solutions)

Now we quantitatively support our concerns that some of the comparisons, mainly the comparison of Lacomme et al. [25] to [21], are not fair, and we aim to give a more fair comparison of these results.

First, we re-evaluated (see Section 3.3.2) the solutions provided by Lacomme et al. [25] using the same travel times as the authors used, i.e., $\check{\tau}$-values, to see if we can reproduce their results. Surprisingly, it turned out that 60 out of these 360 solutions are actually infeasible due to lifespan constraint violation (namely, some batches violate constraint (7)), which not only fundamentally questions the validity of their comparisons, but makes it difficult to make a new, valid one. In 6 further cases, we obtained different values than those indicated by the authors due to their incorrect calculation. In column *L-rev*/$\check{C}_{\max}$ of Table 3 we indicate the average re-evaluated makespan values over the 5×6 solutions for each scenario, including also infeasible solutions. Note that for some instances all the five provided solutions were found to be infeasible, therefore, in fact, if only feasible solutions were taken into account, we would not be able to indicate an average for certain scenarios at all (marked with †).

Then, we re-evaluated these solutions using the same travel times as used in [21], i.e., $\ddot{\tau}$-values, to make a fair comparison. The average values are indicated in column *L-rev*/$\ddot{C}_{\max}$ of Table 3. Note that in 7 cases the provided solution was found to be infeasible even though it was feasible with the other travel times, and thus, one more scenario is marked with †. One of the most important observations we have to make is that the use of different travel times changed the makespan values to a non-negligible extent. Although the increment of the average over all solutions from 7 884 to 7 907 does not seem like much in itself (0.29%), but this alone was enough for Lacomme et al. [25] to be no longer better in this indicator, but worse than Karaoğlan and Kesen [21] with their average value of 7 891.

Despite all this, we should not discount the fact that the solution approach of Lacomme et al. [25] could yield good solutions with a relatively short execution time. Although, for 6 instances all the five provided solutions were infeasible, for 35/8/23 instances the authors provided better/equal/worse solution than Karaoğlan and Kesen [21]. So, there is a reason to assume that by correcting the errors, one would get an indeed efficient solution method.

6.3. Conclusions

The comparison of Karaoğlan and Kesen [21] to [16] is not valid due to the use of different travel times. Since none of the authors provided themselves their best solutions, their methods cannot be compared to each other, unless those experiments are repeated. Note that Karaoğlan and Kesen [21] erred in favor of the others, that is, the results of Karaoğlan and Kesen [21] would probably be improved compared to [16] if the same travel times were used.

The comparison of Lacomme et al. [25] to [21] is not valid, since the authors used different travel times in their experiments. As we showed on the benchmark dataset, the use of travel times rounded in different ways on the one hand could change the set of feasible solutions, and on the other hand it changes the value of the objective function to a non-negligible extent.

There could be some errors in the implementation of the solution approach of Lacomme et al. [25], because the one-sixth of their provided solutions were found to be infeasible due to lifespan constraint violation.

By re-evaluating the accessible solutions of Lacomme et al. [25] with the same travel times as used in [21], we got that the algorithm of Lacomme et al. [25] is not as effective as previously claimed.

7. Evaluation of the local search approach

In this section, we present the results of our computational experiments, where we tested our variable neighborhood search approach on the benchmark dataset. We also compare these results to the revised results from the literature. The purpose of the experiment is to show that our conceptually simple method is able to provide solutions in a relatively short time, that are competitive or even better than the other, sometimes complex solution approaches.

7.1. Setup

We applied our variable neighborhood search (see Section 4.3) for all the 72 instances of the benchmark dataset (see Section 6.1.1). The procedure consisted of 20 rounds. In these experiments we used the original travel times without rounding, i.e., the τ-values.

Table 4
Average makespan values of the best solutions w.r.t. $\check{\tau}$-values.

Q	B	r	G	L	H
300	300	1	10 040	10 032	10 020
300	300	2	9 157	9 144	9 143
300	300	3	9 125	9 116	9 116
300	600	1	10 041	10 018	10 019
300	600	2	9 171	9 144	9 143
300	600	3	9 153	9 119	9 116
600	300	1	8 781	-	8 780
600	300	2	5 744	-	5 265
600	300	3	5 421	-	4 842
600	600	1	8 781	8 780	8 780
600	600	2	5 724	5 284	5 234
600	600	3	5 403	4 856	4 821
average			8 045	-	7 857
avg seconds*			169	413 (83)	317 (16)

[G] Geismar et al. [16].
[L] corrected results of Lacomme et al. [25].
[H] this paper.
* performed in different environments.

Then, the resulted solutions were re-evaluated with $\ddot{\tau}$- and $\check{\tau}$-values for the comparisons. Note that lower bounds were not utilized in these experiments, that is, the solution procedure was not interrupted if the best solution reached the corresponding lower bound indicated in Table 2, i.e., when an optimal solution was found.

Our algorithm was implemented in C++ programming language. All the computational experiments were performed on a workstation with Intel Core i9-7960X (2.80 GHz) processor under Linux operating system using a single thread only.

Note that the algorithms of Geismar et al. [16],[1] Karaoğlan and Kesen [21],[2] Lacomme et al. [25],[3] and Can Atasagun and Karaoğlan [9][4] are not publicly available, and thus their experiments cannot be repeated in our environment. Accordingly, the results provided in the following tables belong tests that performed on different computers. Thus, the execution times, which we indicated only for the sake of completeness, are therefore not comparable, but the qualities of the solutions are still comparable. In the absence of a time limit, we can reasonably assume that the quality of the solutions of Geismar et al. [16] and Lacomme et al. [25] is independent of the running environment. We note however, that the branch-and-cut algorithm of Karaoğlan and Kesen [21] could yield different solution in different environments, especially with a newer version of the MIP solver.

7.2. Comparison to other solution approaches

In Tables 4 and 5, we summarize our results along with the ones known from the literature, while detailed results can be found in the provided supplementary data. In these tables, each of the first 12 rows refers to a scenario (Q, B, r), where the average values of the best solutions of Geismar et al. [16] (G), Karaoğlan and Kesen [21] (KK), Lacomme et al. [25] (L), and ours (H) over the six customer-instances are indicated. The average values over the 72 instances are also indicated. All results were obtained using the same travel times, namely, $\check{\tau}$- and $\ddot{\tau}$-values for Tables 4 and 5, respectively. In Table 4, the indicated values are rounded down as in [16, Table 7]. Recall that we could not collect results into a single table, since the results of Geismar et al. [16] and Karaoğlan and Kesen [21] cannot be compared to each other without repeating any of their experiments.

Note that columns L differ from the appropriate ones in Table 3 for several reasons. First, recall that Lacomme et al. [25] performed five replications for each instance. For the former table, we used the average values of those five solutions, since the authors did the same in their paper, however, for these comparisons we always chose the best solution from the replications. Second, recall that in case of five instances none of the replications yielded feasible solutions, which affected three and four scenarios. While for the former table we also used the values of the infeasible solutions, for this comparison we ignored such solutions, and thus for the affected scenarios no average value could be calculated (indicated with symbol '-').

Based on these tables, our method obtained better solutions in average than the former ones. In Table 5, we also indicate the number of instances where the corresponding approach yielded the best result. As the main outcome, our variable neighborhood search found the best solution in 71 cases, of which in 30 cases it found a better solution than the other two methods. That is, there is only 1 instance, where one of the other approaches found a better solution than our one, and in almost half of the cases, we found a better solution than the other approaches.

[1] BASIC, Windows XP, Intel Pentium 4 (3.2 GHz).
[2] C++ with IBM ILOG CPLEX 12.6 callable library, Windows 7, Intel Xeon (3.16 GHz).
[3] C++, Windows 7, Intel Core i7 (3.4 GHz).
[4] C#, Intel Xeon E-2236 (3.4 GHz).

Table 5
Average makespan values of the best solutions w.r.t. $\breve{r}$-values.

Q	B	r	KK	L	H
300	300	1	10 048.68	10 056.04	10 040.81
300	300	2	9 178.83	9 179.31	9 178.42
300	300	3	9 153.36	-	9 151.76
300	600	1	10 048.14	10 041.86	10 040.81
300	600	2	9 180.05	9 180.12	9 178.42
300	600	3	9 152.15	9 154.65	9 151.76
600	300	1	8 781.54	-	8 781.53
600	300	2	5 345.68	-	5 283.08
600	300	3	4 887.28	-	4 866.77
600	600	1	8 781.54	8 781.53	8 781.53
600	600	2	5 275.67	5 301.69	5 251.84
600	600	3	4 860.75	4 881.36	4 845.31
average			7 891.14	-	7 879.34
best (unique)			10 (1)	40 (0)	71 (30)
avg seconds*			2708	413 (83)	317 (16)

KK Karaoğlan and Kesen [21].
L corrected results of Lacomme et al. [25].
H this paper.
* performed in different environments.

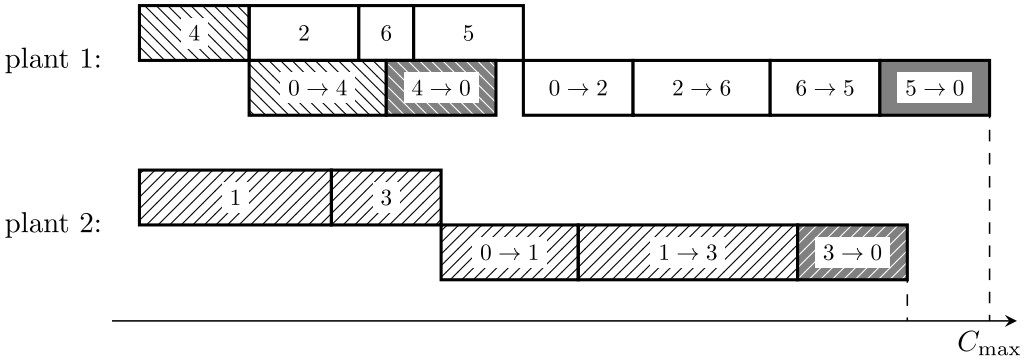

Fig. 5. Example for a solution for the IPODS-MP with 2 plants. Orders 2, 4, 5, and 6 are assigned to the first, orders 1 and 3 are assigned to the second plant.

For the sake of completeness, we also indicate the average running times of the algorithms, although, those were performed on different computers, as we mentioned in the setup. In some respects, it is obvious that the branch-and-cut algorithm of Karaoğlan and Kesen [21] required the longest running time, since their exact method often reached the 1-hour limit. The average running time of the algorithm of Geismar et al. [16] was 169 seconds per instance, the average running time of the algorithm of Lacomme et al. [25] was 413 seconds per instance (that is, 83 seconds per replication), while the average running time of our method was 317 seconds per instance (that is, 16 seconds per round).

7.3. Conclusions

Our conceptually simple local search-based algorithm provides solutions in a relatively short time on the benchmark dataset, that are comparable or even better than the other, sometimes complex solution approaches. That is, there is only 1 out of 72 instance, where one of the other approaches found a better solution than our one, and in almost half of the cases, we found a better solution than the other approaches.

8. Extension to multiple plants

Can Atasagun and Karaoğlan [9] extended the PTSP to multiple plants. In this extension, called IPODS-MP, each plant has a single machine and a single vehicle, which can only transport orders produced at that plant. The production rate (r) is the same for all machines, and the capacity (Q) is the same for all vehicles. Each orders must be first assigned to a plant, then production and transportation are done at each plant in the same way as for the PTSP. The makespan of the solution is the maximum of the makespans of the plants.

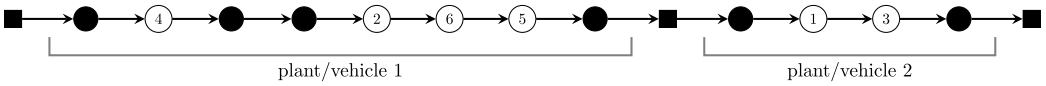

Fig. 6. The canonical representation of the solution depicted in Fig. 5. Separation nodes (black rectangles) separate the vehicles' routes.

Example In Fig. 5, we depict a solution for an instance of the IPODS-MP. The instance is an extension of the instance of Fig. 1 to two plants. Four orders are assigned to the first plant and divided into two batches, these are (4) and $(2, 6, 5)$. The other two orders are assigned to the second plant and form the batch $(1, 3)$. The makespan of the first plant gives the makespan of the entire solution.

8.1. Adaptation of the solution approach

In this section, we briefly describe the adaptation of our variable neighborhood search for the PTSP (see Section 4) to the IPODS-MP.

Modeling approach Similarly as in Section 4.1, we use a route-based solution representation. Instead of using a separate route for each plant, we work with a single route in which we use a new type of nodes, called *separation nodes*, to separate the vehicles. That is, a solution starts and ends with a separation node, and the routes are also separated with a single separation node. A solution is *canonical*, if the batches of the same vehicle are separated with exactly two plant nodes, and no plant node separates an other plant node and a separation node. For an example, see Fig. 6.

Variable neighborhood search The advantage of the single-route solution representation is that we can use the same local search operators as in Section 4.2 without any significant modification, and thus the same variable neighborhood search, which described in Section 4.3.

8.2. Computational experiments

In the following, we present the results of our computational experiments on the benchmark dataset.

8.2.1. Benchmark dataset

Can Atasagun and Karaoğlan [9] created a dataset for the IPODS-MP. The *customer-instances* were generated with different parameter settings, these are the number of customers $(10, 20, 30, 40, 50, 100)$, the number of plants $(2, 3, 4, 5)$, the demand distribution ($[100, 200]$, $[100, 300]$), and the space distribution ($[-100, 100]$, $[-150, 150]$, $[-200, 200]$). Each of these 216 instances was examined with 12 different *scenarios* ($Q \in \{300, 600\}$, $B \in \{300, 600\}$, $r \in \{1, 2, 3\}$) as in case of the PTSP, resulting 2592 *instances* in total.

8.2.2. Setup

We applied our variable neighborhood search (see Section 8.1) for all the 2592 instances of the benchmark dataset. The procedure consisted of 20 rounds. For the large instances (i.e., instances with 100 customers), we also set a 2-hours time limit. In our experiments we used the original travel times without rounding, i.e., the τ-values. Then, the resulted solutions were evaluated with $\ddot{\tau}$-values for the comparisons.

Recall that our algorithm was implemented in C++ programming language. All the computational experiments were performed on a workstation with Intel Core i9-7960X (2.80 GHz) processor under Linux operating system using a single thread only. Note that the variable neighborhood search of Can Atasagun and Karaoğlan [9] was coded using C#, and their tests were performed on a workstation powered by an Intel Xeon (R) E-2236 (3.40 GHz) processor with 32 GB RAM.

8.2.3. Comparison to other solution approaches

In Table 6, we summarize the results of the variable neighborhood search of Can Atasagun and Karaoğlan [9] (*CAK*) and our variable neighborhood search (*H*). This table contains the average makespan values ($\ddot{C}_{\max}$) and execution times in seconds (*time*) on the instances grouped by the number of customers (n). We also indicate the number of instances, where the methods gave the best solutions (*best*), and the number of instances, where this solution is better than the solution of the other method (*unique*).

Note that Can Atasagun and Karaoğlan [9] proposed also a mixed-integer linear programming formulation for the problem, and the authors used GAMS/CPLEX 20.1 to solve instances with the branch-and-cut procedure. However, only 68 instances were (proven) solved to optimality within the 2-hours time limit, and for 411 instances no feasible solution was found. In addition, the MIP solver found a better solution than the variables neighborhood search in only 81 cases. Thus, we exclude this procedure from the comparison.

The variable neighborhood search of Can Atasagun and Karaoğlan [9] slightly outperformed our method on the small instances (i.e., instances with 10 customers). That is, method CAK gave 0.6% better makespan values on average than method H, and obtained better/equal/worse results on 31/171/14 instances. On medium instances (i.e., instance with 20-50 customers), our variable neighborhood search was better. Although, the average improvement is only 0.8% in average, method H obtained better/equal/worse results on 979/458/291 instances. On the large instances (i.e., instances with 100 customers), the variable neighborhood search of Can Atasagun and Karaoğlan [9] again outperformed our method. The improvement is again relatively small (1.6%), but method CAK obtained

Table 6
Comparison of solution approaches for the IPODS-MP.

	n	CAK		H	
		$\bar{C}_{\max}$	time	$\bar{C}_{\max}$	time
Small	10	**837.39**	0.09	842.81	0.44
	best (unique)	202 (31)		185 (14)	
Medium	20	1 267.79	0.48	**1 258.83**	14.36
	30	1 816.89	3.86	**1 803.44**	117.21
	40	2 339.06	20.51	**2 321.24**	476.81
	50	2 863.61	75.20	**2 841.00**	1 201.77
	average	2 071.84	25.01	**2 056.13**	452.54
	best (unique)	749 (291)		1437 (979)	
Large	100	**3 379.45**	212.31	3 433.48	7 211.95
	best (unique)	510 (436)		212 (138)	
All	average	**2 295.87**	69.76	2 299.36	2 104.71
	best (unique)	1461 (758)		1834 (1131)	

[CAK] variable neighborhood search of Can Atasagun and Karaoğlan [9].
[H] variable neighborhood search of this paper.

better/equal/worse results on 436/74/138 instances. Overall, our variable neighborhood search obtained better/equal/worse results on 1131/703/758 instances, however, method CAK resulted in an average 0.2% better makespan on the full dataset.

The reason our method was not so effective on the large instances is that the variable neighborhood search was slow in case of instances with 100 customers. The 2-hours time limit was reached in all cases, and thus not all the 20 rounds were performed. On the small and medium instances, the average execution times are also higher than for method CAK, however, in case of method H, the execution times refer to 20 rounds.

8.3. Conclusions

Our adapted variable neighborhood search is competitive in terms of solution quality with the existing solution approach for the IPODS-MP on the benchmark dateset. For 1131 out of 2592 instances, we obtained better solutions than the method of Can Atasagun and Karaoğlan [9], and for 703 instances our method obtained solutions with the same makespan value. However, our method has a significantly longer execution time.

9. Conclusions

In this paper, we investigated an integrated production and outbound distribution scheduling problem, where a single product with a short lifespan, a single machine, and a single vehicle are considered. We considered both the single-plant (PTSP) and the multi-plant (IPODS-MP) variants. We reviewed the existing results of the PTSP, and demonstrated that most of the comparisons of these results are not valid. By re-evaluating the accessible solutions, we provided a more fair comparison of the existing methods. We proposed a variable neighborhood search for the problem, and our computational experiments showed the this conceptually simple, elementary method outperforms the former, sometimes complex solution approaches on the widely-used benchmark dataset. That is, there is only 1 instance out of 72, where one of the other approaches found a better solution than our one, and in almost half of the cases, we found a better solution than the other approaches. We adapted our approach to the extension of the problem to multiple plants. Our computational experiments showed that this adaption is competitive in terms of solution quality with the existing solution approach for the problem, as our method found a better solution than the former approach for almost half of the instances.

The PTSP can be extended in several other points, for example, multiple machines, and multiple vehicles may be considered. Our solution approach could be also adjusted to these extensions. The dynamic variant of the problem also promises to be interesting. That is, orders are not known in advance but come in an online fashion, thus the production needs to be re-optimized.

CRediT authorship contribution statement

Markó Horváth: Conceptualization, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary data

The benchmark datasets used for the computational experiments (see Section 6.1.1 and Section 8.2.1), the obtained solutions, and the implementation of the evaluation procedure (see Section 3.3.2) can be found under https://github.com/hmarko89/PTSP-supplementary. Detailed computational results can be found in the provided supplementary data.

Acknowledgements

This research has been supported by the TKP2021-NKTA-01 NRDIO grant on “Research on cooperative production and logistics systems to support a competitive and sustainable economy”. The work of the author was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. The author is grateful to his colleagues, Tamás Kis and Péter Györgyi, for their constructive comments that helped to improve the paper significantly.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ejco.2024.100095.

References

[1] Y. Adulyasak, J.F. Cordeau, R. Jans, The production routing problem: a review of formulations and solution algorithms, Comput. Oper. Res. 55 (2015) 141–152.
[2] P. Amorim, M. Belo-Filho, F.D. Toledo, C. Almeder, B. Almada-Lobo, Lot sizing versus batching in the production and distribution planning of perishable goods, Int. J. Prod. Econ. 146 (2013) 208–218.
[3] R. Armstrong, S. Gao, L. Lei, A zero-inventory production and distribution problem with a fixed customer sequence, Ann. Oper. Res. 159 (2008) 395–414.
[4] W.J. Bell, L.M. Dalberto, M.L. Fisher, A.J. Greenfield, R. Jaikumar, P. Kedia, R.G. Mack, P.J. Prutzman, Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer, Interfaces 13 (1983) 4–23.
[5] M. Belo-Filho, P. Amorim, B. Almada-Lobo, An adaptive large neighbourhood search for the operational integrated production and distribution problem of perishable products, Int. J. Prod. Res. 53 (2015) 6040–6058.
[6] L. Berghman, Y. Kergosien, J.C. Billaut, A review on integrated scheduling and outbound vehicle routing problems, Eur. J. Oper. Res. (2023).
[7] L. Bertazzi, M.G. Speranza, Inventory routing problems: an introduction, EURO J. Transp. Logist. 1 (2012) 307–326.
[8] G. Can Atasagun, İ. Karaoğlan, Integrated production and transportation scheduling problem with multiple plants, multiple vehicles and perishable products, in: International Conference on Management Science and Engineering Management, Springer, 2022, pp. 616–628.
[9] G. Can Atasagun, İ. Karaoğlan, Solution approaches for the integrated production and outbound distribution scheduling problem with multiple plants and perishable items, Expert Syst. Appl. 237 (2024) 121318.
[10] H.K. Chen, C.F. Hsueh, M.S. Chang, Production scheduling and vehicle routing with time windows for perishable food products, Comput. Oper. Res. 36 (2009) 2311–2319.
[11] Z.L. Chen, Integrated production and outbound distribution scheduling: review and extensions, Oper. Res. 58 (2010) 130–148.
[12] L.C. Coelho, J.F. Cordeau, G. Laporte, Thirty years of inventory routing, Transp. Sci. 48 (2014) 1–19.
[13] P. Devapriya, Optimal fleet size of an integrated production and distribution scheduling problem for a single perishable product, Ph.D. thesis, Clemson University, 2008.
[14] P. Devapriya, W. Ferrell, N. Geismar, Integrated production and distribution scheduling with a perishable product, Eur. J. Oper. Res. 259 (2017) 906–916.
[15] P. Farahani, M. Grunow, H.O. Günther, Integrated production and distribution planning for perishable food products, Flex. Serv. Manuf. J. 24 (2012) 28–51.
[16] H.N. Geismar, G. Laporte, L. Lei, C. Sriskandarajah, The integrated production and transportation scheduling problem for a product with a short lifespan, INFORMS J. Comput. 20 (2008) 21–33.
[17] P.C. Gilmore, R.E. Gomory, Sequencing a one state-variable machine: a solvable case of the traveling salesman problem, Oper. Res. 12 (1964) 655–679.
[18] C.H. Glock, E.H. Grosse, J.M. Ries, The lot sizing problem: a tertiary study, Int. J. Prod. Econ. 155 (2014) 39–51.
[19] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, in: Annals of Discrete Mathematics, vol. 5, Elsevier, 1979, pp. 287–326.
[20] F. Jafari Nozar, J. Behnamian, Hyper-heuristic for integrated due-window scheduling and vehicle routing problem for perishable products considering production quality, Eng. Optim. 53 (2021) 1902–1921.
[21] İ. Karaoğlan, S.E. Kesen, The coordinated production and transportation scheduling problem with a time-sensitive product: a branch-and-cut algorithm, Int. J. Prod. Res. 55 (2017) 536–557.
[22] Y. Kergosien, M. Gendreau, J.C. Billaut, A benders decomposition-based heuristic for a production and outbound distribution scheduling problem with strict delivery constraints, Eur. J. Oper. Res. 262 (2017) 287–298.
[23] P. Lacomme, A. Moukrim, A. Quilliot, M. Vinot, PTSP results, https://perso.isima.fr/~lacomme/marina/Research/PTSP_MIM.html, 2015. (Accessed 9 January 2024).
[24] P. Lacomme, A. Moukrim, A. Quilliot, M. Vinot, The integrated production and transportation scheduling problem based on a GRASP×ELS resolution scheme, IFAC-PapersOnLine 49 (2016) 1466–1471.
[25] P. Lacomme, A. Moukrim, A. Quilliot, M. Vinot, Supply chain optimisation with both production and transportation integration: multiple vehicles for a single perishable product, Int. J. Prod. Res. 56 (2018) 4313–4336.
[26] J. Lee, B.I. Kim, A.L. Johnson, K. Lee, The nuclear medicine production and delivery problem, Eur. J. Oper. Res. 236 (2014) 461–472.
[27] J.Y.T. Leung, Z.L. Chen, Integrated production and distribution with fixed delivery departure dates, Oper. Res. Lett. 41 (2013) 290–293.
[28] K. Li, Z.h. Jia, J.Y.T. Leung, Integrated production and delivery on parallel batching machines, Eur. J. Oper. Res. 247 (2015) 755–763.
[29] L. Liu, S. Liu, Integrated production and distribution problem of perishable products with a minimum total order weighted delivery time, Mathematics 8 (2020) 146.
[30] F. Marandi, S. Zegordi, Integrated production and distribution scheduling for perishable products, Sci. Iran. 24 (2017) 2105–2118.
[31] C. Meinecke, B. Scholz-Reiter, A representation scheme for integrated production and outbound distribution models, Int. J. Inf. Syst. Logist. Manag. 18 (2014) 283–301.
[32] S. Moons, K. Ramaekers, A. Caris, Y. Arda, Integrating production scheduling and vehicle routing decisions at the operational decision level: a review and discussion, Comput. Ind. Eng. 104 (2017) 224–245.
[33] R. Russell, W.C. Chiang, D. Zepeda, Integrating multi-product production and distribution in newspaper logistics, Comput. Oper. Res. 35 (2008) 1576–1588.

[34] H. Shaabani, A literature review of the perishable inventory routing problem, Asian J. Shipp. Logist. 38 (2022) 143–161.
[35] C. Viergutz, Integrated production and distribution scheduling, Ph.D. thesis, University of Osnabrück, 2011.
[36] C. Viergutz, S. Knust, Integrated production and distribution scheduling with lifespan constraints, Ann. Oper. Res. 213 (2014) 293–318.
[37] D.Y. Wang, O. Grunder, A.E. Moudni, Integrated scheduling of production and distribution operations: a review, Int. J. Ind. Syst. Eng. 19 (2015) 94–122.