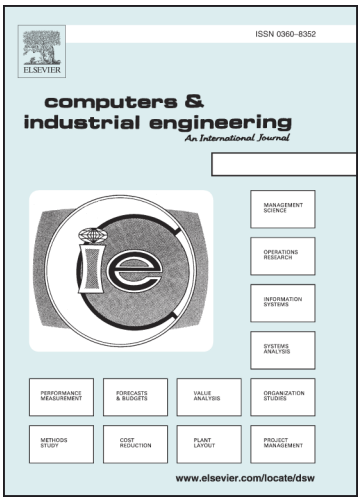


Synchronizing vehicles for multi-vehicle and one-cargo transportation

Zhi-Hua Hu, Chen Wei

PII: S0360-8352(18)30108-6
DOI: https://doi.org/10.1016/j.cie.2018.03.023
Reference: CAIE 5127

To appear in: *Computers & Industrial Engineering*

Received Date: 10 October 2015
Revised Date: 15 November 2017
Accepted Date: 14 March 2018

Please cite this article as: Hu, Z-H., Wei, C., Synchronizing vehicles for multi-vehicle and one-cargo transportation, *Computers & Industrial Engineering* (2018), doi: https://doi.org/10.1016/j.cie.2018.03.023

Synchronizing vehicles for multi-vehicle and one-cargo transportation

Zhi-Hua Hu[1*], Chen Wei[1]

[1]Logistics Research Center, Shanghai Maritime University, Shanghai 200135, China

[*]Please address all correspondence to:

Zhi-Hua Hu, Ph. D.

Logistics Research Center

Shanghai Maritime University

Shanghai 200135, China

Tel: 86-21-38284016

Fax: 86-21-58855200

Email: zhhu@shmtu.edu.cn

Synchronizing vehicles for multi-vehicle and one-cargo transportation

Abstract

In ship manufacturing and big-size cargo transportation, several flat vehicles are usually cooperated to transport one big-size ship segment or cargo, namely "multi-vehicle and one-cargo transportation" (MVOC). It is distinctly different from general transportation scenarios where a vehicle is used to load and transport several cargos, namely "one-vehicle and multi-cargo transportation" (OVMC). The MVOC generates the difficulty in synchronizing several vehicles for transporting a cargo. A mathematical program that can be solved by existing mixed-integer linear program solver is formulated, by considering the synchronization constraint among the unit-load flat vehicles under the minimization of makespan. To improve the computation performance of the solution methods, a sequential insert algorithm is developed as the basic procedure for a greedy insert algorithm and as the decoding scheme of a proposed genetic algorithm. These three methods (the mathematical program, the greedy insert algorithm and the genetic algorithm) are compared by numerical studies considering the effects of instance scales and synchronization complexity on the optimality and computation performances. The genetic algorithm performed better for solving small- and medium-scale instances with lower complexity, and the greedy insert algorithm is very fast and suitable for solving large-scale and complex instances.

Keywords

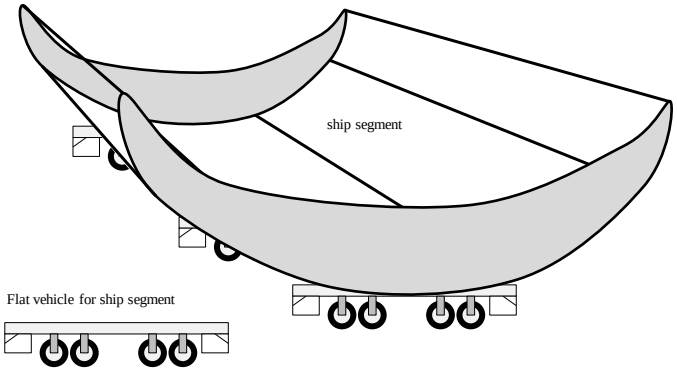
Logistics management; vehicle routing problem; big-size cargo transportation; synchronization; genetic algorithm

1. Introduction

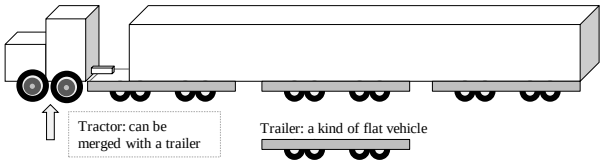
The following two scenarios of synchronizing flat vehicles activated this study. First, in the ship manufacturing, the ship segments with various shapes and sizes are transported by several coordinated and synchronized flat vehicles. Usually, a ship segment is shipped by at least two flat vehicles. Second, in the big-size cargo transportation, generally a tractor is coordinated with multiple synchronized trailers. We named these vehicles as unit-load vehicles that can be synchronized and connected for transporting various sizes cargos. In general logistics and transportation problems, a vehicle can visit multiple costumers to serve the customers' demands, , and this mode is named as "one-vehicle and multi-cargo transportation" (OVMC). Significantly different from OVMC, several unit-load vehicles are synchronized when transport a big-size cargo, namely "multi-vehicle and one-cargo transportation" (MVOC). Considering the ship manufacturing and big-size cargo scenarios above, the unit-load vehicles are "flat" because there is a flat platform on the top of the driving cabs, as depicted in Figure 1.

Two typical MVOC scenarios are described below. In a ship manufacturing factory, a ship is manufactured and assembled by segments gradually. Although the technologies and segments for building ships are somewhat standardized, the shapes and sizes of the segments are various in the manufacturing processes. In the ship manufacturing factory, these segments are stored, handled and transported according to the process flows of the ship design. Therefore the segment logistics is important for organizing the process flows. Generally, several flat vehicles (Figure 1(a)) are used to load a segment cooperatively. The flat vehicles have two driving cabs under the platform and can move to any directions by rotating the wheels. To transport a big-size segment, the number of used vehicles is determined by the weight and size of the segments, and a tractor is used cargo on roads,

as presented in Figure 1(b). The big-size cargo transportation problem therefore can be viewed as a MVOC problem coupled with the tractor assignment problem.



(a) Flat vehicles for transporting ship segments in ship manufacturing factory



(b) Big-size cargo transportation

Figure 1. Two typical MVOC scenarios

As a distinct feature of the two scenarios above, handling a transportation task requires several synchronized vehicles. A demonstration of the handling is presented in Figure 2. In the left-upper corner of Figure 2, one -unit cargo is transported by a single unit-load vehicle, while 2 units or 2.5 units cargos need to be transported by two or three synchronized vehicles. In the network in the right of Figure 2, the task c1 is directly transported by a unit-load vehicle from the position w2 to w4, while the unit-load task c2 is transported by a unit-load vehicle v2 from w3 to w5; after completing task c1 and c2, v1 and v2 meet at w1; then a bi-unit-load cargo c3 is handled by synchronizing v1 and v2.

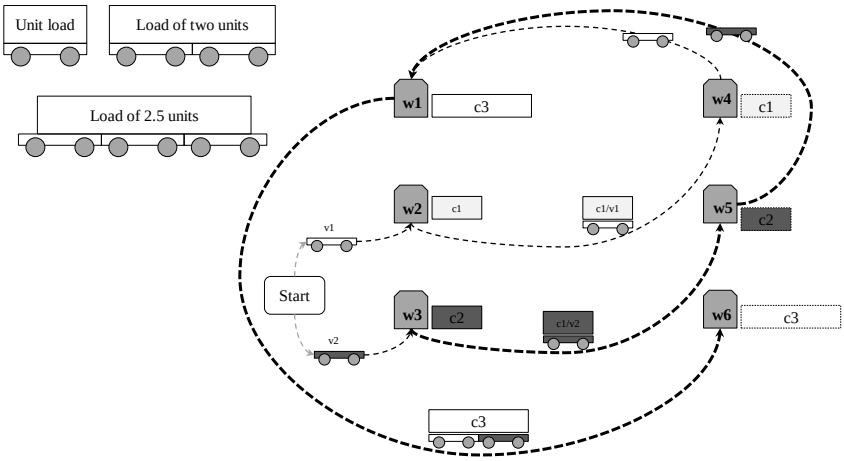


Figure 2. A demonstration of synchronizing vehicles in MVOC

The basic form of the MVOC problem can be defined as follows. A set of unit-load vehicles with available times and positions, and a set of multi-load transportation tasks with origin and destination positions are given. The problem is to assign the vehicles for each task to minimize

the makespan. As an important synchronous constraint, a *n*-load task must be handled by synchronizing *n* vehicles. The synchronization problem of these unit-load flat vehicles for transporting one cargo includes the following three distinct features. First, the vehicles are homogeneous. Second, a cargo transportation task requires several unit-load vehicles. Third, for a task, all required vehicles must be coordinated and synchronized before departing the task position, and can be rescheduled only after completing the task. General solutions for vehicle routing problems (VRP) were usually developed for solving OVMC and cannot be directly applied for MVOC.

Activated by the typical MVOC scenarios, this study contributes to the literature in the following aspects. First, the synchronization of unit-load flat vehicles is proposed in the background of ship manufacturing and big-size cargo transportation. Although the synchronization in OVMC problems is formulated in literature, to the best of our knowledge, it is the first time to consider the synchronization in MVOC. Second, because the algorithms for OVMC cannot be applied to MVOC, we contribute to developing new algorithms. An algorithm based on sequential insert algorithm is developed by considering the synchronization of available vehicles when a task is inserted to the task lists of vehicles. The vehicles are assigned to a task based on their available times and the minimal makespan of the schedule when the vehicles are synchronized for a task. A greedy insert (GI) algorithm and generic algorithm (GA) are developed based on this sequential insert algorithm. Third, besides the algorithm parameters and performances, the effects of synchronization parameters on the solutions are examined.

The rest sections are organized as follows. After reviewing the related studies on OVMC and MVOC (Section 2), a formal mathematical program is devised to minimize the makespan of conducting a set of tasks (Section 3). In Section 4, inspired by experiences of the operators in ship manufacturing environments, a sequential insert algorithm is developed by inserting the tasks into the schedule sequentially according to the priorities of the tasks; based on the sequential insert algorithm, an GI algorithm is proposed by inserting a task from the unscheduled tasks into the schedule under the minimization of makespan at each step; then, a GA is designed by using the sequential insert algorithm as the decoding scheme. In Section 5, the synchronization constraints are analyzed by numerical studies and the three algorithms are compared in terms of optimality and computation times. The managerial and technological implications are also discussed. Finally, the study is concluded in Section 6.

2. Related studies

2.1. OVMC transportation

The well-known travelling salesman problem (TSP) and VRP are typical OVMC problems. They focus on optimizing the route to traverse a set of customers by a vehicle or several vehicles. These two categories of routing problems have many applications in industry with the variants. For example, Kim *et al.* (2015) surveyed the City VRP literature categorized by stakeholders and summarized the constraints, models, and solution methods for VRP in urban cities. Similar to city logistics, City VRP mainly differs from VRP in terms of the involved stakeholders, namely the shipper, carrier, resident and administrator. Montoya-Torres *et al.* (2015) presented a state-of-the-art survey on the VRP with multiple depots. In the review, several variants of the VRP model are studied: time windows, pickup and delivery, split delivery, periodic deliveries, and heterogeneous fleet. The review classified the approaches according to whether the optimal

objective is single or multiple. Pillac *et al.* (2013) classified routing problems from the perspective of information quality and evolution. After a general description of dynamic routing, they introduced the concept of dynamism degree and presented a comprehensive review of applications and solution methods for dynamic VRPs .

The truck-and-trailer and inter-mode drayage transportation problems are seemly related to this study. However, these two categorized transportation problems are all consistent with the fact that a vehicle/trailer serves a set of customers by transporting cargos for these customers. In the truck-and-trailer system (Derigs *et al.*, 2013; Lin *et al.*, 2011; Villegas *et al.*, 2013), several trailers are sequentially connected and driven by a tractor as the engine. In the drayage transportation mode (Marković *et al.*, 2014; Xue *et al.*, 2014; Zhang *et al.*, 2014), the tractor can be detached from a container trailer, so that the tractor can be efficiently utilized.

2.2. Heavy and big-size cargo transportation

As studied in the Section 1, the big-size cargo transportation and logistics in the shipbuilding yards activated this study. The flat vehicles are used to deliver heavy blocks from one plant to another in shipyards. Park and Seo (2012) addressed the transporter scheduling and routing problem at a shipyard, which was transformed into parallel machine scheduling with sequence-dependent setup times and precedence constraints. The GRASP algorithm was developed for transporter scheduling and routing. Joo and Kim (2014) considered a scheduling problem of block transportation with a delivery restriction to determine when and by which transporter each block is delivered. A mathematical model for the optimal solution was derived, and two algorithms based on GA and a self-evolution algorithm (SEA) were proposed. These two papers handled the transportation of the heavy ship segments and the routing problem of the flat vehicles. However, the fact that a cargo may be loaded by several vehicles symultenously was not forumulated.

2.3. Routing with synchronization

As a distinct feature of MVOC, a cargo is simultaneously loaded by several vehicles, therefore, synchronizing these vehicles is a critical operation. Dohn *et al.* (2011) and Drexl (2012) pointed out that synchronization problems are highly relevant in routing practice.

The temporal intra-route and inter-route synchronizations are typical in literature. Ioachim *et al.* (1999) used the concept of synchronization in routing for the fleet assignment problem and extended to general temporal dependencies by Dohn et al. (2011). Gschwind (2015) formulated the synchronized pickup and delivery problem as a prototypical VRP with temporal intra-route synchronization in the sense that synchronization occurred only within disjunctive pairs of nodes. Generally, temporal intra-route synchronization can be explained by the vehicle's visiting order at customers. Comparing to intra-route synchronization, inter-route synchronization is not widely researched. Salazar-Aguilar *et al.* (2012) introduced a synchronized arc routing problem for snow plowing operations. The street segments with two or more lanes in the same direction are plowed simultaneously by synchronized vehicles. Salazar-Aguilar *et al.* (2013) introduced the paint vehicle synchronization problem that several capacitated vehicles painted lines on the roads with a tank vehicle replenishing the painting vehicles. The routes and schedules for the painting and tank vehicles were optimized by synchronizing the painting and replenishment operations. Rousseau *et al.* (2013) solved a real-time vehicle dispatching problem where some customers were serviced with multiple resources synchronously. Derigs and Pullmann (2014) introduced the multi-depot multi-trip VRP with order incompatibilities subjected to inter-route synchronization constraints.

In the MVOC transportation, a vehicle cooperates with other vehicles to transport heavy

and big-size cargos. The cooperation of vehicles causes the inter-route synchronization problem. When two vehicles load the same cargo, their routes should be synchronized for the transportation. This synchronization is deduced by the demand of sharing the same route segment among routes for transporting the same cargo.

The VRP with cross-docking (VRPCD) and the VRP with trailer and transshipments (VRPTT) are also constrained by inter-route synchronization constraints (Drexl, 2012). In the VRPCD, loading or unloading a vehicle depends directly on the consolidation and the pickup/delivery route plans. Morais *et al.* (2014) developed a constructive heuristic and six local search procedures based on general VRP literature for solving the VRPCDs. Solutions of pickup and delivery with cross docking were also studied by Wen *et al.* (2009) and Chen *et al.* (2006). Cortes *et al.* (2010) considered a VRPCD problem with a limited number of transfer points where the vehicles are synchronized. In the VRPTT each vehicle is composed by a truck and trailers. Considering the customers' accessibility, the trailer can be detached at satellites while the truck serves the customers' demand alone. For a survey on formulations and algorithms to VRPTT, we refer to Drexl (2012) and Drexl (2013). In these two categories of synchronization constraints, generally the cargo is transferred from a vehicle (truck, tractor or trailer) to another vehicle. Therefore, they are different from the MVOC investigated in this study.

2.4. Summary

Three streams of research are categorized based on the above reviews and listed in Table 1. Based on the features as presented and analyzed, the MVOC proposed in this study is explicitly different from OVMC that is widely identified and studied in literature. Although the difficult constraints in MVOC can be explained as synchronization among vehicles, the present researches on temporal (mainly intra-route and inter-route) and resource synchronizations do not reflect the synchronization among several vehicles for transporting a single cargo.

Table 1. Related routing problems and features

Category	Reference	Features
OVMC	Kim et al. (2015) Montoya-Torres et al. (2015) Pillac et al. (2013)	In the VRP and TSP solutions, each vehicle carries and delivers cargos to several customers from a depot.
	Derigs et al. (2013) Villegas et al. (2013) Lin et al. (2011)	In truck-and-trailer systems, the cargos for several customers in a route are loaded by several trailers which can be detached in the route.
	Marković et al. (2014) Xue et al. (2014) Zhang et al. (2014)	In the drayage transportation mode, the tractor can be detached from a container trailer, so the tractor can be efficiently utilized when it is used to a sequence pickup an delivery tasks.
MVOC	Park and Seo (2012)	The transporter scheduling and routing problem at a shipyard, which can be transformed into parallel machine scheduling with sequence-dependent

		setup times and precedence constraints.
	Joo and Kim (2014)	A scheduling problem of block transportation under a delivery restriction to determine when and by which transporter each block is delivered from its source plant to destination plant.
Synchronization	Dohn et al. (2011) Drexl (2012) Ioachim et al. (1999) Gschwind (2015) Derigs and Pullmann (2014)	The temporal intra-route and inter-route synchronizations are typical in VRPs.
	Salazar-Aguilar et al. (2012)	Two vehicles are synchronized in arcs in arc routing problems.
	Rousseau et al. (2013)	A customer must be served by several vehicles, which are subject to synchronization constraints.

3. Formulation

To establish a formal mathematical program, the sets and indices, known data and decision variables are denoted in the following. Notably, the set and known data are represented by capital letters, while the indices and variables are lower cases.

Two sets, tasks and vehicles, are involved. I is the set of cargo transportation tasks $(1,2,...,N)$, indexed by i and j; V is the set of unit-load vehicles indexed by v. To facilitate the formulations, two dummy tasks (*S* and *E*) are added to I, and denote $I^+ = I \cup \{S, E\}$.

The tasks data are given as follows. The pickup position of task i is denoted by $P_i = \left(P_i^X, P_i^Y\right)$, where $\left(P_i^X, P_i^Y\right)$ are the position coordinates. Similarly, the delivery position of task i is denoted by $D_i = \left(D_i^X, D_i^Y\right)$. These position data are used for computing the transportation time of tasks and the travel times between any two tasks. The available time for handling the task i is denoted by A_i. Ω_i is the handling time of task i, which is represented by the transportation time of a vehicle from the pickup position P_i to the delivery position D_i for the task i. When the Euclidean distance is used, $\Omega_i = \|D_i - P_i\| / \Theta$, where Θ is the vehicle speed. The speed is set to a constant representing the average speed of vehicles for simplifying the formulations, ignoring the speed should be different for vehicles with or without heavy loads. The number of unit loads, the same as the number of unit-load vehicles synchronized for the task i, is denoted by L_i. The travel time from the delivery position of task i to the pickup position of task j is denoted by T_{ij}, which is computed by $T_{ij} = \|D_i - P_j\| / \Theta$. Notably, we use the task-based indices for the routing problem.

As for a vehicle *v*, $U_v = \left(R_v^X, R_v^Y\right)$ denotes the initial position of the vehicle v at the vehicle available time R_v. Because the vehicle should move to the pickup position to handle a task, we use R_{iv} to represent the available time of the vehicle v for handling task i, namely $R_{iv} = \|U_v - P_i\| / \Theta + R_v$. In a summary, R_{iv}, T_{ij} and Ω_i can be computed based on P_i, D_i, U_v and Θ as follows: $R_{iv} = \|U_v - P_i\| / \Theta + R_v$; $T_{ij} = \|D_i - P_j\| / \Theta$; and $\Omega_i = \|D_i - P_i\| / \Theta$.

The assignments of tasks to vehicles and the times of conducting the tasks are optimized

in this study. Three sets of decision variables are involved. The binary variable x_{ijv} is formulated as that if the tasks i and j are handled by vehicle v sequentially, x_{ijv} is one; otherwise, zero. Based on x_{ijv}, binary variable y_{iv} presents that if the task i is conducted by vehicle v, y_{iv} is one; otherwise, zero. Besides these assignment and sequencing relations, z_i is used to denote the finish time of task i.

The objective and constraints are given in [M1]. The Constraints (1) and (2) minimize the makespan. The Constraints (3) sets the low bounds of the tasks according to the task available times, while the Constraints (4) guarantees that the task start only after the assigned vehicle is available at the task's pickup position. In Constraints (5), the start time of a task is restricted by the finish time of precedent task operated by the same vehicle (x_{ijv}). The vehicle can travel from the precedent task delivery position to the next task's pickup position only after finishing the precedent task. In the Constraints (6) and (7), the assignment variable (y_{iv}) is presented by the sequencing variable (x_{ijv}). In the solution, each vehicle has a route that connects a set of tasks sequentially, which is strengthened by Constraint (8). A valid route starts at a dummy task (*S*) (Constraints (9)) and ends at another dummy task (*E*) (Constraints (10)), and the sub circles in the route is eliminated by Constraints (5) and (11). The Constraint (12) is synchronization constraint that L_i vehicles are assigned to the task i simultaneously. These synchronization constraints make [M1] prominently different from the formulations of typical routing and scheduling problems. Constraints (13) and (14) guarantee that the variables are integral.

[M1] min f (1)

s.t.

$$f \geq z_i, \forall i \in I \quad (2)$$

$$z_i \geq A_i, \forall i \in I \quad (3)$$

$$z_i + (1 - y_{iv}) M_1 \geq R_{iv} + \Omega_i, \forall i \in I, v \in V \quad (4)$$

$$z_j + (1 - x_{ijv}) M_2 \geq z_i + T_{ij} + \Omega_j, \forall i, j \in I, i \neq j, v \in V \quad (5)$$

$$\sum_{j \in I^+ \setminus i} x_{ijv} = y_{iv}, \forall i \in I, v \in V \quad (6)$$

$$\sum_{j \in I^+ \setminus i} x_{jiv} = y_{iv}, \forall i \in I, v \in V \quad (7)$$

$$x_{ijv} + x_{jiv} \leq 1, \forall i, j \in I, i \neq j, v \in V \quad (8)$$

$$\sum_{j \in I^+ \setminus S} x_{Sjv} = 1, \forall v \in V \quad (9)$$

$$\sum_{i \in I^+ \setminus E} x_{iEv} = 1, \forall v \in V \quad (10)$$

$$x_{iiv} = x_{iSv} = x_{Eiv} = 0, \forall v \in V, i \in I^+ \quad (11)$$

$$\sum_v y_{iv} = L_i, \forall i \in I \quad (12)$$

$$x_{ijv}, y_{iv} \in \{0,1\}, \forall i, j \in I^+, v \in V \quad (13)$$

$$f \geq 0, z_i \geq 0, \forall i \in I^+ \quad (14)$$

In Constraints (4) and (5), two big numbers M_1 and M_2 are involved for formulating the real numbers (z_i) by the integer values (y_{iv} and x_{ijv}). The big numbers affect the searching process

and optimality of general mixed-integer linear program (MILP) solvers. Typically, they can be determined by the maximal values of the right hand expressions in (4) and (5), as $M_1 \geq \sum_{iv} R_{iv} + \sum_i \Omega_i$ and $M_2 \geq M_1 + \sum_{ij} T_{ij} + \sum_i \Omega_i$. By adding the synchronization constraints, [M1] extends the general VRP model. The experimental study (Section 5) demonstrates that the MILP solvers (e.g., Gurobi and Cplex) cannot solve even small instances within acceptable times. The VRP solution algorithms in literature fail to consider Constraints (12) and can not be used directly to solve [M1]. In Section 4, algorithms are developed for solving [M1] efficiently.

4. Solution algorithms

Three algorithms are developed to solve the MVOC problem. First, the sequential insert algorithm is devised based on the operational experience from industry (Section 4.1). According to the interviews at the Shanghai Waigaoqiao Shipbuilding Co., Ltd. (cn.chinasws.com), the operators use the task priority to assign the multi-load transportation tasks to vehicles in a planning period. When inserting a task into the schedule, the feasibility and synchronization constraint is guaranteed under the minimization of makespan. The interview and observation is a direct inspiration of developing an algorithm by using sequential insert strategy (see Section 4.1). Second, the GI algorithm based on the sequential insert strategy is developed to generate improved solutions (see Section 4.2). Third, based on the sequential insert algorithm that provides a chromosome representation scheme, a GA is developed in Section 4.3.

4.1. Sequential insert algorithm

Figure 3 illustrates the process of inserting tasks into the schedule. For simplicity, the handling time of each task (T_i) is set to one time unit, and the travel time (T_{ij}) between two tasks is determined by $T_{ij} = |i - j|$. In the upper part of Figure 3, a sequence of tasks is given (we take the sequence as a chromosome in the GA, which is studied in Section 4.3). Sixteen tasks and three bi-load tasks (#4, #8 and #10 represents the 4th, 8th and 16th task) are involved. As shown in the lower part of Figure 3, three vehicles are used. The tasks and vehicles are initially available. The first three one-load tasks (#1, #2 and #3) are assigned to the vehicles #1, #2 and #3 individually. The vehicles handle these tasks at time 0 and finish the tasks at time 1. Because the task #4 is a bi-load task, two vehicles are synchronized for transporting task #4. Considering the travel time for each vehicle, vehicle #1, #2 and #3 can start to handle task #4 at time 4, 3 and 2. To minimize the makespan of the four tasks #1 to #4, the vehicles #2 and #3 should be synchronized at time 3 for task #4, and the four tasks can be finished at time 4. Similarly, the rest tasks can be inserted into the schedule sequentially. In Figure 3, four numbers around a scheduled task (e.g., #16) are explained: '16' is the task number; '20' is the start time of task #16 handled by vehicle #1, while '21' is the task finish time; '17' is the finish time of the previous task of task #16 in vehicle #1's task sequence. The setup time between task #16 and #13 is 3, equals to the travel time (T_{ij}) from task #13 to #16.

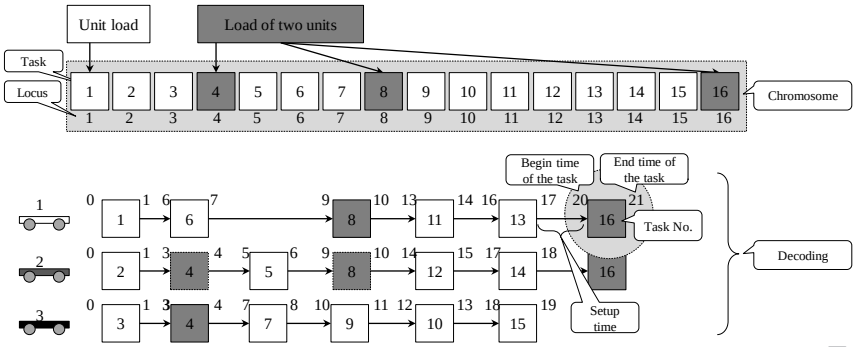


Figure 3. A illustration of the sequential insert algorithm

The procedures described in Figure 3 are formulated as the Algorithm 1.

Algorithm 1. Sequential insert algorithm

Inputs 1) $[I, V, \Theta, R_{iv}, T_{ij}, T_i, L_i, A_i, P_i, D_i]$;

2) p : task sequence, whose k-th task is accessed by $p(k)$

Outputs 1) r_v : a route represented by the task sequence of the vehicle v;

2) $[s_i, e_i]$: the start and finish times of the task i.

Steps

Variables 1) Define e_v^I and e_v^T as the last task and finish time of vehicle v.

2) Define e_v^{T*} as the temporary finish time of vehicle v

Step 1 1) Initialize: for each $v \in V$, set $r_v = [\ ], e_v^I \leftarrow 0$, $e_v^T \leftarrow 0$;

2) Initialize: for each $i \in I$, set $[s_i, e_i] = 0$;

Step 2 For each $p(k)$, $k = 1, 2, \cdots, |p|$

Step 2.1 Initialize: $i \leftarrow p(k)$; $e_v^{T*} \leftarrow 0, \forall v$.

Step 2.2
$$e_v^{T*} = \begin{cases} \max(A_i, R_{iv}) + T_i, e_v^V = 0 \\ e_v^T + T_{e_v^I, k} + T_k, e_v^V > 0 \end{cases}, \quad \forall v \in V$$

Step 2.3 Find vehicle set V^* ($|V^*| = L_i$), where $e_{v^*}^{T*} \le e_v^{T*}$ for all $v^* \in V^*$ and $v \in V \setminus V^*$.

Step 2.4 $t \leftarrow \max\{e_{v^*}^{T*} : v^* \in V^*\}$

Step 2.5 $r_{v^*} \leftarrow [r_{v^*}, i], e_{v^*}^I \leftarrow i, e_{v^*}^T \leftarrow t, \forall v^* \in V^*$

Step 2.6 $[s_i, e_i] = [t - T_i, t]$

End for

The sequential insert algorithm takes task sequence (p) and known data defined in Section 3 as input parameters, assigns the tasks to vehicles (r_v) and schedules the start and finish

times ($[s_i, e_i]$) for the tasks. e_v^I, e_v^T and e_v^{T*} refer to the last task, finish time and temporary finish time of vehicle v. In Step 1, the outputs (r_v and $[s_i, e_i]$) are initialized as empty vectors and zeros. Then, for each task (i) in p, assign the task to the vehicles and computing temporary finish time e_v^{T*} (Step 2.1 and Step 2.2). Subsequently, choose L_i vehicles synchronized for the task i (Step 2.3). Finally, update the output values by the assignment of tasks to vehicles (Steps 2.4, 2.5 and 2.6).

4.2. Greedy insert algorithm

The Algorithm 1 is inspired by operational experiences and works when the priorities of the tasks are strictly given as p. Based on the principles of choosing candidate vehicles to be synchronized for the multi-load tasks (Step 2 of Algorithm 1), the GI algorithm is developed as the Algorithm 2. In the Algorithm 1, the task sequence is given and the tasks are inserted into the schedule according to the task order in the sequence. In Algorithm 2, the sequence is not given and for every step a task is chosen under the minimization of makespan at the current step. The step 2 of the Algorithm 2 is devised similarly to the step 2 of the Algorithm 1.

Algorithm 2. Greedy insert algorithm

Inputs $[I, V, \Theta, R_{iv}, T_{ij}, T_i, L_i, A_i, O_i, D_i, T_{a,b}^P]$;

Outputs 1) r_v: a route represented by the task sequence of the vehicle v;

2) $[s_i, e_i]$: the start and finish times of the task i

Variables

1) Define e_v^I and e_v^T as the last task and finish time of vehicle v.

2) Define e_v^{T*} as the temporary finish time of vehicle v.

3) Define e_k^T as the finish time of the k-th task in I when the task is chosen to insert to the vehicle's task sequence under the minimal makespan.

4) Define e_k^V as the coordinated vehicles for the k-th task in I when the task is chosen to insert to the vehicle's task sequence under the minimal makespan.

Steps

Step 1 1) Initialize: $e_k^T \leftarrow 0, e_k^V \leftarrow [\ ], \forall k \in \{1, 2, \cdots, |I|\}$

2) Initialize: for each $v \in V$, set $r_v = [\ ], e_v^I \leftarrow 0$, $e_v^T \leftarrow 0$;

3) Initialize: for each $i \in I$, set $[s_i, e_i] = 0$;

Step 2 For each $k \in \{1, 2, \cdots, |I|\}$

Step 2.1 Initialize: $i \leftarrow I(k)$; $e_v^{T*} \leftarrow 0, \forall v$.

Step 2.2 $e_v^{T*} = \begin{cases} \max\left(A_i, R_{iv}\right) + T_i, e_v^V = 0 \\ e_v^T + T_{e_v^I,k} + T_k, e_v^V > 0 \end{cases}$, $\forall v \in V$

Step 2.3 Find vehicle set V^* ($\left|V^*\right| = L_i$), where $e_{v^*}^{T*} \le e_v^{T*}$ for all $v^* \in V^*$ and $v \in V \setminus V^*$.

Step 2.4 $t \leftarrow \max\left\{e_{v^*}^{T*} : v^* \in V^*\right\}$

Step 2.5 $e_k^T \leftarrow t, e_k^V \leftarrow V^*$

End for

Step 3 $k^* = \underset{k}{\operatorname{argmin}}\left\{e_k^T\right\}$, $V^* \leftarrow e_{k^*}^V$, $i \leftarrow I(k^*)$

Step 4 $r_{v^*} \leftarrow [r_{v^*}, i], e_{v^*}^I \leftarrow i, e_{v^*}^T \leftarrow t, \forall v^* \in V^*$

Step 5 $[s_i, e_i] = [t - T_i, t]$

Step 6 Remove the k-th element in I: $I(k) = [\,]$

Step 7 If I is not empty, Go to Step 1.

The procedure of the Algorithm 2 is described as follows. In Step 1, the variables are initialized. Then, in Step 2, a loop is used to compute the makespan of each task by assuming the task is inserted into the schedule. In Step 3 and 4, choosing the task with minimal makespan and inserting it into the schedule. In Step 5, the start and finish times of the chosen task are updated. The set of unscheduled tasks is updated in Step 6.

4.3. The genetic algorithm

The algorithms proposed in Sections 4.1 and 4.2 can find a solution fast while the optimality cannot be guaranteed. Therefore, a GA is developed in this section.

GA is a popular solution method, often used to optimize the solutions for problems in many fields, such as engineering, computer science, economic management, and supply chain management based on natural selection that drives biological evolution (Dao *et al.*, 2017). The GA repeatedly modifies a population of chromosomes (individual solutions) to improve the solution. During a generation, the GA selects chromosomes randomly from the current population as parents and uses the parents to generate the children as the next generation. Over successive generations, the population evolves toward an optimal solution. The GA uses three main types of rules at each generation to create the next generation from the current population (Karakatič and Podgorelec, 2015): 1) By selection rules, the parents are selected to generate the population at the next generation; 2) By crossover rules, two parents are combined to form children for the next generation; 3) By mutation rules, random changes are applied to parents to form children.

As presented in the Algorithm 3, the GA works as follows (Karakatič and Podgorelec, 2015). The algorithm begins by initializing a population randomly, and then creates a sequence of new populations. At a generation, the algorithm uses the individuals (chromosomes) to create the next population. Generally, the new population is created by the following steps. First, evaluate each

chromosome of the current population by computing the fitness value via a fitness function. Second, scale the raw fitness scores by converting them into a more usable range of values. Third, select some chromosomes as parents based on their fitness. Some individuals with better fitness in the current population are chosen as the elites that are passed to the next population. Fourth, generate the children from the parents. Children are produced either by making random changes to a single parent (mutation) or by combining the chromosomes from a pair of parents (crossover). Fifth, replace the current population with the children to form the next generation. The algorithm stops when the termination criterion is met.

Algorithm 3. The GA

Begin

$t \leftarrow 0$;

Generate the initial population $P(t)$;

Evaluate $P(t)$ and select the elite solutions $E(t)$;

While (termination criteria are not satisfied) **do**

Begin

Crossover $P(t)$ to generate $C^X(t)$;

Mutate $P(t)$ to generate $C^M(t)$;

Merge $C^X(t)$ and $C^M(t)$ as $C(t)$ and Evaluate $C(t)$;

Move best solutions from $P(t)$ and $C(t)$ to $E(t)$;

Select $P(t+1)$ from $P(t)$ and $C(t)$ by roulette wheel selection;

$t \leftarrow t+1$;

End

end

4.3.1. Permutation-based representation

To run a metaheuristic, the first step is to determine an encoding scheme. In GA, this step is known as the chromosome representation, which is the most important problem of designing a GA. In this study, the representation of a chromosome uses a permutation of the list of tasks. As demonstrated in Figure 1, the representation can decode any given permutation into a feasible solution according to the Algorithm 1.

4.3.2. Initialization

The algorithm begins by creating a random initial population that contains chromosomes generated by the permutation-based method.

4.3.3. Fitness evaluation

Corresponding to an optimization model formulated as regular mathematical programs, the fitness function is the objective function that needs to optimize. When the regular mathematical program is a minimization one, the fitness of a solution should be as lower as possible. A solution is related to a chromosome, the chromosome's decoding form, and the decision variables in the mathematical program. The GA generally use processed fitness function in the selection scheme. The raw fitness values returned by the fitness function are converted into the values in a range that is suitable for the selection scheme, entitled as fitness scaling. The selection function uses the scaled fitness values to select the parents for the next generation by assigning a higher probability of selection to individuals with higher scaled values. The range of the scaled values affects the performance of the GA. If the scaled values vary too widely, the individuals with the highest scaled

values reproduce too rapidly, taking over the population gene pool too quickly, and preventing the GA from searching other areas of the solution space. On the other hand, if the scaled values vary only a little, all individuals have approximately the same chance of reproduction and the search will progress very slowly. In this study, the rank-based scaling method (Stern *et al.*, 2006) is used, which scales the raw scores based on the rank of each chromosome instead of its score. The rank of an individual is its position in the sorted fitness values: the rank of the fittest chromosome is 1, the rank of the next fittest is 2, and so on. The scaled value of a chromosome with rank n is proportional to $\sqrt{n}$. The rank-based fitness scaling removes the effect of the spread of the raw scores.

4.3.4. Crossover and mutation

At each generation, the GA uses the current population to create the children that make up the next generation by reproduction methods. The algorithm selects a group of chromosomes in the current population, called parents, who contribute their genes for their children. The algorithm usually selects chromosomes that have better fitness values as parents by the selection scheme.

The GA creates three types of children for the next generation. First, elite children are the chromosomes in the current generation with the best fitness values. These individuals survive to the next generation. Second, crossover children are created by combining the genes from a pair of parents. Third, mutation children are created by introducing random changes, or mutations, to a single parent. Both crossover and mutation processes are essential to the GA. Crossover enables the algorithm to extract the best genes from different individuals and recombine them into potentially superior children. Mutation adds to the diversity of a population and thereby increases the likelihood that the algorithm will generate individuals with better fitness values. Three parameters are used to control the three types of children reproduction methods.

1) The elite count (P^E) specifies the number of chromosomes that are used to guaranteed the best chromosomes survive to the next generation. The elite count is a positive integer less than or equal to the population size.

2) The crossover fraction (P^X) specifies the fraction of the next generation, other than elite children, that are produced by crossover. The crossover fraction is a real number between 0 and 1. Based on $P(t)$, three sets of chromosomes should be generated for the next generation: $E(t)$, $C^X(t)$ and $C^M(t)$, with the sizes of P^E, $\left\lfloor \left(|P(t)| - P^E\right) \cdot P^X \right\rfloor$, and $\left(|P(t)| - P^E\right) - \left\lfloor \left(|P(t)| - P^E\right) \cdot P^X \right\rfloor$ individually.

3) Mutation probability (P^M) is used to control the diversity of the mutation method. A higher value of P^M introduces more variances to the parent when generate children.

In the following, the crossover and mutation methods in the proposed GA are described.

(1) The crossover method

Crossover is responsible for exchanging genetic features between selected parents. Position-based crossover (Syswerda, 1990) is applied to the berthing priority vector. It is essentially a kind of uniform crossover for permutation representation. The procedure is defined in Algorithm 4 and illustrated in Figure 4.

Algorithm 4. Position-based Crossover (PX)

Input two parents

Output offspring

Step 1 Select a set of positions from one parent at random.

Step 2 Produce a proto-child by copying the nodes on these positions into the corresponding positions of it.

Step 3. Delete the nodes which are already selected from the second parent. The resulted sequence of nodes contains the nodes the proto-child needs

Step 4. Place the nodes into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce one offspring.

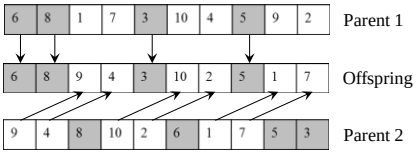


Figure 4. Illustration of PX crossover operator

(2) Mutation method

Inversion mutation is used to mutate the berthing priority vector. Inversion mutation selects two random positions within a chromosome and then inverts the substring between these two positions, as illustrated in Figure 5. The procedure described in Figure 5 is denoted by $p \leftarrow Inverse(p,i,j)$, where p in the right is the chromosome before mutation, i and $j(j>i)$ are two positions of the segment to be inversed, and the p in the left is the mutated chromosome. Based on the inversion, Algorithm 5 is developed as the mutation method.

Figure 5. Illustration of inversion mutation operator

Algorithm 5. Inverse-based Mutation

Input A chromosome (parent, p), the mutation probability (P^M)

Output A mutated chromosome (offspring, p)

Step 1 $i \leftarrow 1$.

Step 2 **While** $i < |p|$

Step 2.1 **If** $rand() > P^M$, **continue**.

Step 2.2 Choose j from $(i,|p|]$ randomly.

Step 2.3 $p \leftarrow Inverse(p,i,j)$

End While

The above crossover and mutation operators always produce feasible solutions because the generated children are always permutations of the task sequence.

4.3.5. Elite reservation strategy

In the GA, a certain number of elite solutions (the ratio is named as elite count, denoted by P^E as described in Section 4.3.4) are reserved in an extra archive and updated at every generation according to the fitness values of the solutions (the fitness is computed by the procedure in Section 4.3.3).

4.3.6. Selection

The selection methods choose parents for the next generation based on the scaled values from the fitness scaling method. Roulette selection is used to select the parents by simulating a roulette wheel, in which the area of the section of the wheel corresponding to an individual is proportional to the individual's expectation (Goldberg, 1989). The algorithm uses a random number to select one of the sections with a probability equal to the section area size.

5. Numerical results

Three solution methods (namely [M1], GI and GA) are tested in this section. [M1] is solved by Gurobi 6.0 (www.gurobi.com); the GI (Algorithms 2) and GA (Algorithm 3) are implemented by Matlab and the experiments are performed on the Intel(R)-64 Core(TM) 2 CPU, 6600@2.4 GHz with 2 GB of memory.

5.1. Dataset

The test datasets are encoded as "*IxVy(z1,z2,…,zy)*", where "*x*" is the number of tasks, "*y*" is the number of vehicles, and "*zi*" is the ratio (a number in [0,1]) representing the number of tasks that require "*i*" vehicles cooperated to handle them. Three series of datasets are used in the experiments, as shown in Table 2. The first is a detailed small-scale instance with 20 tasks (Table A.1). The numbers of one-unit, bi-unit and tri-unit load tasks are 8, 5 and 7 individually. In the second dataset, three types of cargos are considered and their numbers are the same (Table A.2). The number of tasks is set to 100. In the third series of datasets, the statistical distribution of L_i is generated according to the specific experimental purposes (Table A.3).

Table 2. Dataset settings

No.	Datasets	Settings
1	I20V10(8,7,7)	P_i^X, D_i^X and R_v^X are generated from the uniform distribution U[1,5400]; P_i^Y, D_i^Y and R_v^Y are generated from U[1,1000]; L_i is randomly set to 1, 2 or 3; A_i is generated from U[1,60] and R_v is generated from U[1,60]. The data is presented in Table A.1 (see the Appendix).
2	I100V10(1/3,1/3,1/3)	P_i^X, D_i^X, R_v^X, P_i^Y, D_i^Y, R_v^Y, A_i and R_v are generated as the Dataset 1. However, a set of coordinates (P_i^X, D_i^X) are generated as shown in Table A.2. Next, the origin and destination for each task are chosen from this set. The data of tasks are shown in Table A.3. Then,

		the available position of each vehicle is also chosen from Table A.2, and the available position and time of each vehicle are listed in Table A.4.
3	I100V10(.)	P_i^X, D_i^X, R_v^X, P_i^Y, D_i^Y, R_v^Y, A_i and R_v are generated as the Dataset 2, while the value of L_i is set in the experiment.

Note: U[a,b] is an uniform distribution with the two bounds, namely *a* and *b*.

5.2. Experimental settings

The basic parameters of the GA are set by Table 3. When the GA is tested, the algorithm ran 50 times for each problem generally. These parameters are determined by the following steps. First, the above Dataset 2 is used for tests. Second, the population size, crossover faction and mutation probability are set by the values in [20, 25, 30, 35, 40, 45, 50], [0.5, 0.6, 0.7, 0.8, 0.9] and [0.01, 0.02, …, 0.1] individually for combination experiments. Third, when the GA cannot find an improved solution within ten generations, the algorithm is terminated and the value of average generations is about 50. Therefore, we set the maximal generation to 100 in the following tests. Fourth, for each combination, 50 times of run were executed. The optimal solutions under the parameter combinations were compared. The best combinations were presented as Table 3 for default parameter settings of the GA. Notably, the mutation probability did also vary from 0.1 to 0.2 and the results were not apparently improved.

Table 3. Default parameter settings of the GA.

Parameters	Values
Population size	40
Maximal generation	100
Crossover fraction	0.7
Mutation probability	0.1
Elite count	2
Termination criterion	The number of maximal generations is reached.

In the following, the MVOC routing problem is studied by using three methods: [M1] that is directly solved by a MILP solver (Gurobi 6.0, www.gurobi.com), the GI algorithm (Algorithm 2), and the GA (Algorithm 3). The experiments primarily aim at assessing these algorithms when the synchronization constraints are set. Table 4 summarizes five experimental scenarios, corresponding model adjustments, experimental steps and final results in tables or figures.

Table 4. Settings of computational experiments

No.	Purpose	Settings	Results
1	Demonstrate the results of the three algorithms	1) use dataset 1	Figure 6.
2	Study the effects of crossover fraction and mutation probability on solutions	1) use dataset 1; 2) the crossover faction (P^X) is set to 0, 0.1, …, or 1; 3) the mutation probability (P^M) is set to	Figure 7

		0, 0.1, …, or 1; 4) for each P^X and P^M, the GA is run for fifty times; 5) the crossing effects of P^X and P^M on solution quality are depicted by contour plots.	
3	Compare optimality and computation time of the three algorithms	1) use dataset 2; 2) Increase the number of tasks ($\|N\|$) from 10 to 100 by 5 for each time; 3) set the time limit of the solver for solving [M1] to nine hours; 4) for each $\|N\|$, use the MILP solver, GI algorithm and the GA to solve the instances; 5) compare the optimality and computation times.	Figure 8
4	Examine the effect of ratio of bi-load tasks (of only one-load and bi-load tasks) on the solutions	1) use dataset 3; 2) increase the ratio of bi-load tasks from 0.1 to 1 by 0.1 for each time; 3) for each ratio, randomly choose a set of tasks according to this ratio, and set their values of L_i to 2; 4) use the GI algorithm to solve the problem; 5) use the GA to solve the instances for fifty times and record the optimums; 6) examine the average optimums for each ratio setting.	Figure 9
5	Examine the effect of distribution of L_i on the solutions	1) use datasets 1 and 3; 2) set L_i by an uniform distribution $U[1,a]$ where $a=1,2,\cdots,10$; 3) for dataset 1 and each a, use the MILP solver and GA to solve the instances; 4) for dataset 3 and each a, use the GA to solve the instance.	Figure 10

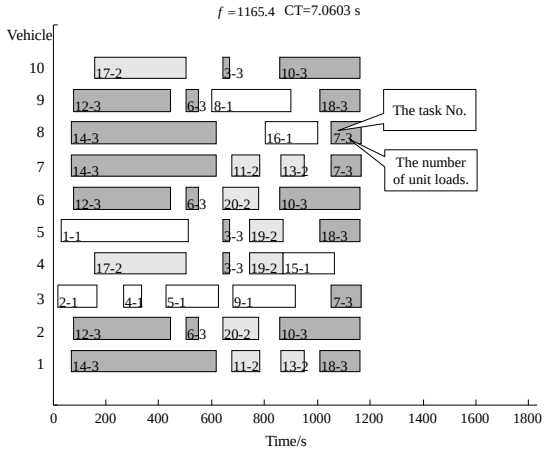
The results of the experiments described in Table 4 are analyzed in the section 5.3.

5.3. Results

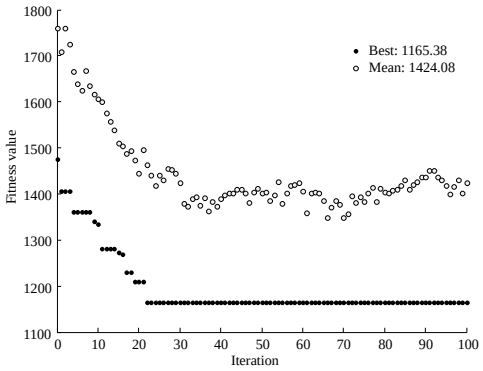
5.3.1. Demonstration of the three solution methods

The MILP solver (solving [M1]), the GI algorithm and the GA are compared for solving an instance with twenty tasks and ten vehicles. The numbers of one-, bi- and tri-load tasks are set in the dataset. In Figure 6, the assignment and timing results are presented by diagrams. As presented by the Figure 6(a) and Figure 6(b), the GA get the best result (1165.38) within about two seconds

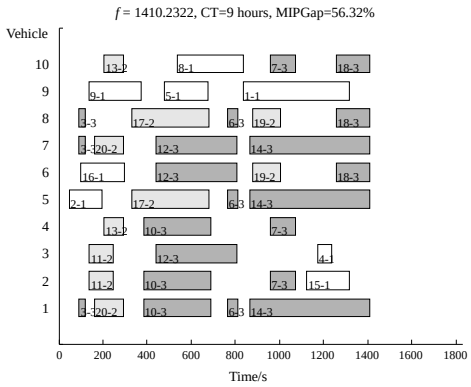
(the GA got the best fitness value after 25 iterations). When [M1] was solved by the MILP solver (Gurobi), even after nine hours, the MIPGap was still 56.32% and the optimal result is 1410.23 (about 1.25 times of the result of the GA) (Figure 6(c)). The GI is very fast and can return the makespan within far less than one second (Figure 6(d)). However, its solution value is 1.57 times of the result of the GA. The results are summarized in Table 5.



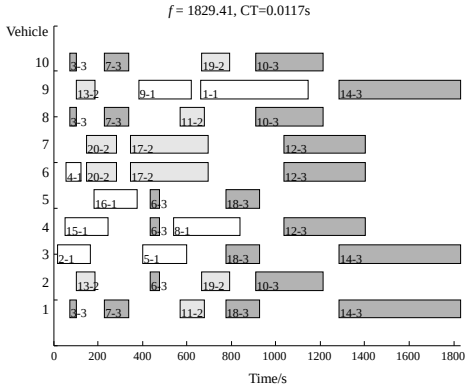
(a) Results of GA



(b) Fitness of GA



(c) Results of solving [M1]



(d) Results of the GI

Figure 6. A demonstration of the GA, [M1] and GI methods (No.1)

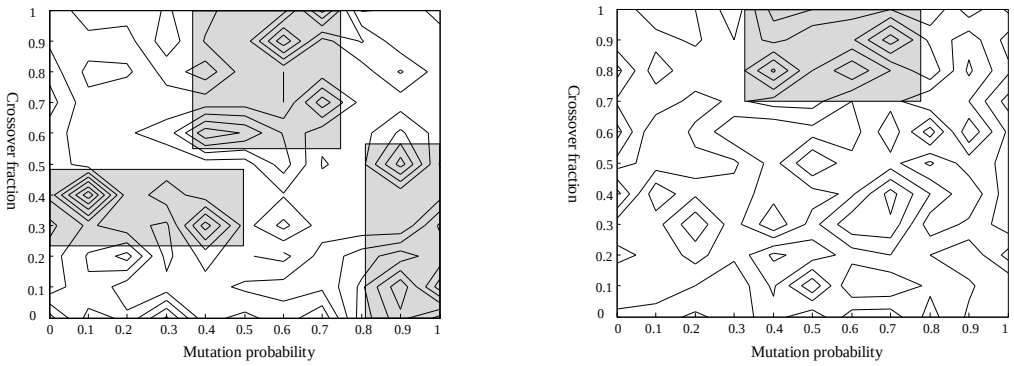
Table 5. Demonstrative results of optimality and computing time (No. 1)

Solution method	Objective	Computing time
MILP solver	1410.23	9 hours
GI algorithm	1829.41	0.0117 seconds
GA	1165.40	7.0603 seconds

5.3.2. Effects of crossover fraction and mutation probability on solutions

The crossing experiments are conducted to test the crossover fraction and the mutation probability. For each combination of these two algorithmic parameters, the GA is performed for fifty runs. The minimal and mean values of the makespan are depicted by contour plots (Figure 7). In Figure 7(a), three blocks are identified, while one block is identified in Figure 7(b). As defined in the section 6.5, the number of chromosomes generated by crossover and mutation operators are controlled by the crossover fraction, while the degree of mutation is controlled by the mutation

probability.



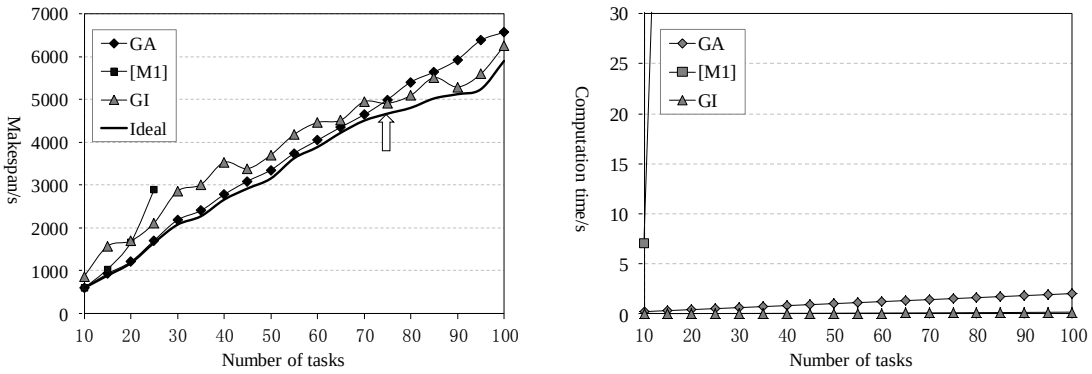
(a) min (b) mean

Figure 7. Tuning the crossover fraction and mutation probability (No.2)

5.3.3. Optimality and computation time of the three algorithms

The optimality and computation time are affected by the number of tasks, which is examined in Figure 8. To compare the optimality, for each setting of the number of tasks, the GA ran 1000 times and the best solution is taken as the ideal lower bound (denoted as "Ideal"), depicted as a thick line in Figure 8(a). As shown in the two figures in Figure 8, the MILP solver can only solve instances with less than 30 tasks. Therefore, we mainly compare the solutions from GA and GI. As for computation time, the GI is absolutely competitive (Figure 8(b)). In the aspect of optimality, the GA is competitive comparing to the GI when the number of tasks is less than 80. Then, for the instances of more than 80 tasks, the GI shows advantage.

Table 6 presents the comparison results of the three solution methods on the optimality and computing performance. The gaps between the three solution methods and the ideal solution are computed. The MILP solver can obtain the optimal solution only for the smallest instance and can obtain some feasible solutions within nine hours for the instances of 10-25 tasks. The GA is competitive when the tasks are about less than 70, while the GI algorithm is competitive when more tasks are involved in the instances. Additionally, the GI is always efficient to find a feasible solution (less than 0.2 seconds).



(a) Optimality (b) Computation time

Figure 8. Comparing performances of the GA, [M1] and GI methods (No.3)

Table 6. Optimality and computation time of the three solution methods (No.3)

Tasks	Optimality							Computing time/Second		
	Ideal	[M1]	Gap	GI	Gap	GA	Gap	[M1]	GI	GA
10	592	592	**0.00%**	865	46.11%	593	**0.17%**	7	0.0292	6.78
15	883	1033	16.99%	1568	77.58%	958	**8.49%**	32400	0.0047	8.2
20	1174	1657	41.14%	1693	44.21%	1237	**5.37%**	32400	0.0075	9.38
25	1651	2891	75.11%	2112	27.92%	1767	**7.03%**	32400	0.0120	10.5
30	2071	-	-	2852	37.71%	2259	**9.08%**	-	0.0164	10.52
35	2267	-	-	3007	32.64%	2493	**9.97%**	-	0.0221	11.32
40	2656	-	-	3530	32.91%	2918	**9.86%**	-	0.0280	12.74
45	2916	-	-	3381	15.95%	3225	**10.60%**	-	0.0321	13.64
50	3149	-	-	3699	17.47%	3488	**10.77%**	-	0.0399	15.48
55	3622	-	-	4178	15.35%	3955	**9.19%**	-	0.0483	16.76
60	3884	-	-	4458	14.78%	4281	**10.22%**	-	0.0558	18.24
65	4219	-	-	4512	**6.94%**	4556	7.99%	-	0.0642	18.96
70	4501	-	-	4945	**9.86%**	4872	**8.24%**	-	0.0736	19.34
75	4665	-	-	4905	**5.14%**	5214	11.77%	-	0.0831	19.7
80	4801	-	-	5090	**6.02%**	5650	17.68%	-	0.0960	21.62
85	5020	-	-	5506	**9.68%**	5911	17.75%	-	0.1058	23.12
90	5120	-	-	5287	**3.26%**	6239	21.86%	-	0.1197	21.5
95	5233	-	-	5590	**6.82%**	6569	25.53%	-	0.1332	24.58
100	5896	-	-	6245	**5.92%**	6877	16.64%	-	0.1470	22.76

Note: '-' means that no solution is found; Gap(x)=(x-Ideal)/Ideal.

5.3.4. Effect of ratio of bi-load tasks on the solutions

The ratio of the bi-load tasks should represent the complexity of the problem. In Figure 9, the optimality degrees of GA and GI are compared. Notably, for each setting of the ratio, the types of the (one-load or bi-load) tasks are randomly settled. In Figure 9, the GA achieved better results than the GI according to the mean and minimal optimums. The ratio near 0.5 generates a wider range of the standard variances, which can be explained that equal single-vehicle and bi-vehicle tasks increase the optimization complexity.

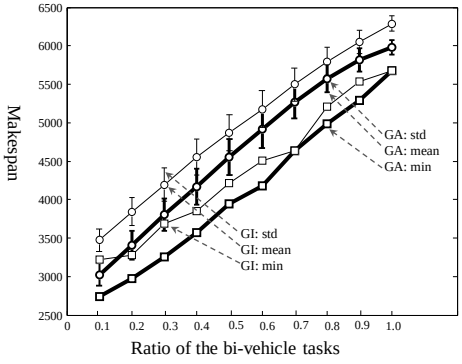


Figure 9. The effect of ratio of bi-vehicle tasks on the solutions (No.4)

5.3.5. Effect of distribution of L_i on the solutions

This experiment is used to test the complexity of the instances on the solutions. Here, the complexity is represented by the variety of the cargos, namely the number of values of L_i. In Figure 10, the complexity is simulated by increasing the upper boundary parameter of the uniform distribution whose lower bound is one. As presented in Figure 10, from left to right, the complexity increases. As the general trends, the makespan and variance values increase. Comparing the GA and GI, the GI performs better especially when the complexity is higher.

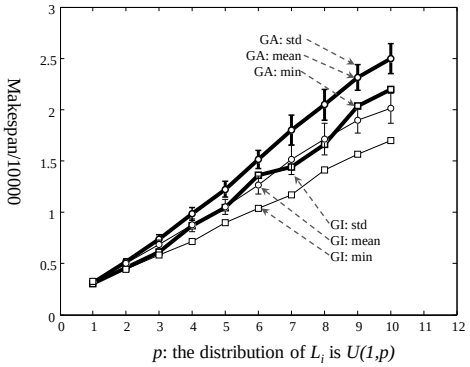


Figure 10. The effect of ratio of distribution of L_i on the solutions (No.5)

5.4. Discussions

Based on the numerical results and analyses of the five experiments, the managerial implications are discussed below.

1) The GA can solve small instances optimally within acceptable computation time comparing to the MILP solver and the GI. Although the validity of the formulations is verified by the MILP solver, it is unsuitable for practical applications because it cannot yield competitive solutions. The GI is very fast while its optimality cannot be guaranteed.

2) The crossover fraction and the mutation probability are two important parameters for tuning the performances of the GA. Generally, these two parameters impose less significant effects on the solutions. However, when we can pursue the optimality by running the GA repeatedly, e.g., applying parallel computing technologies to solve the problem, the combination of these two parameters should be finely tuned.

3) The GI is a fast algorithm, while the GA can also obtain better solutions within acceptable time spans for small- and medium-scale instances. For large-scale instances, the GI is competitive in optimality and computation time. Therefore, when the problem can be solved by rolling-horizon strategy, the GA can be used because the scales of the number of tasks may be reduced to an ideal number suitable for the GA. Otherwise, the GI is recommended.

4) The GA also differs from the GI in the complexity of the problems. For simple instances, the GA performs better than the GI (Experiment 4, Figure 9), whereas the GI is better for complex instances (Experiment 5, Figure 10). Therefore, when a decision-support system is developed, the GA and GI both should be implemented because they can solve instances of different complex degrees.

6. Conclusions

Activated by application scenarios in the fields of ship manufacturing and big-size cargo

transportation, a special transportation problem where unit-load flat vehicles are synchronized to transport multi-load cargos is examined. To handle the synchronization constraint, a mixed-integer linear program that can be solved by existing solvers is devised by introducing synchronization constraints into routing models. Inspired by experienced operators, a sequential insert algorithm is proposed by sequentially inserting tasks into the schedule under the minimization of makespan. Based on this algorithm, a GI algorithm is developed. The tasks are chosen to be inserted into the schedule by considering the minimization of makespan at each step. To achieve the optimality purpose, a GA is developed by using the sequential insert algorithm as the decoding scheme. By five sets of experiments, these three solution methods are compared in terms of optimality and computation time by the instances with different scales and complexity of the synchronization constraints. Although the mathematical program can be used to guarantee the optimality, it cannot be solved for even small-scale instances. The GA and GI have different advantages according to the numerical results. The GA can find better solutions within acceptable computation times for small- and medium-scale instances, and the instances with lower complexity. However, the GI is absolutely competitive in computation time, and is suitable for optimizing large-scale and complex instances. These results can be used for decisions on choosing methods according to the features of the instances.

Based on the formulations and developed methods, the following practical features can be further studied to improve the applicability of the algorithms. First, we consider a batch of tasks whose available times are not differentiated. The algorithms should be extended to support the time windows or precedence-dependent orders of the tasks. Second, a static scheduling problem is examined, whereas the handling and travel times are uncertain. Therefore, dynamic scheduling methods could be developed based on the proposed algorithm. Further, advanced computing technologies, e.g., parallel computing and cloud computing, can be used to improve the optimality within limited time.

Acknowledgements

This study is partially supported by the National Nature Science of China (71471109), the Science and Technology Commission of Shanghai (16040501800), and the ministry of transport of the People's Republic of China (2015329810260).

Appendix A. Tables

Table A.1. A test instance with tasks and vehicles (dataset 1, *I20V10(8,7,7)*)

Task							Vehicle			
No.	P_i		D_i		L_i	A_i	No.	U_v		R_v
	P_i^X	P_i^Y	D_i^X	D_i^Y				R_v^X	R_v^Y	
1	5160	336	1175	698	1	0	1	864	203	8
2	5125	196	3972	622	1	0	2	3709	577	9
3	1163	208	1263	5	3	0	3	5178	79	2

4	3160	898	2626	788	1	0	4	1737	673	6
5	2607	6	3964	933	1	0	5	5144	516	8
6	645	427	266	444	3	0	6	3379	493	2
7	2203	394	1292	560	3	0	7	473	408	6
8	629	697	3105	852	1	0	8	1158	280	9
9	3709	577	1770	203	1	0	9	2617	307	0
10	2696	518	181	287	3	0	10	1770	203	1
11	4685	119	3783	140	2	0				
12	3223	517	181	287	3	0				
13	3385	659	2662	668	2	0				
14	653	202	5178	79	3	0				
15	1657	298	65	540	1	0				
16	3709	577	2092	827	1	0				
17	2903	236	14	127	2	0				
18	2800	187	1747	886	3	0				
19	647	7	1657	298	2	0				
20	951	79	2024	353	2	0				

Table A.2. Coordinates of points used to set the origins and destinations of tasks (Datasets 2 and 3)

No.	X	Y	No.	X	Y	No.	X	Y	No.	X	Y	No.	X	Y
1	3223	517	21	2100	60	41	4184	756	61	2646	908	81	4857	992
2	308	862	22	3379	493	42	3753	474	62	2473	739	82	230	408
3	4806	679	23	4150	521	43	4045	12	63	4617	661	83	3516	884
4	5379	274	24	4597	110	44	1115	954	64	1770	203	84	1836	630
5	2203	394	25	5109	748	45	2290	654	65	3972	622	85	4673	924
6	2662	668	26	4181	151	46	3555	553	66	65	540	86	1292	560
7	3755	815	27	645	427	47	5125	196	67	5089	339	87	2447	418
8	2184	863	28	2626	788	48	266	444	68	4935	200	88	4144	293
9	3105	852	29	1737	673	49	1175	698	69	1975	639	89	5178	79
10	181	287	30	2024	353	50	1163	208	70	4635	488	90	1989	340
11	1965	187	31	2450	836	51	2607	6	71	4101	76	91	2092	827
12	3606	97	32	1263	5	52	1747	886	72	2270	970	92	3412	917
13	4687	518	33	493	106	53	2800	187	73	2824	533	93	3783	140
14	4235	530	34	1524	406	54	4659	288	74	1659	600	94	951	79
15	3147	387	35	1913	50	55	3964	933	75	3338	234	95	1242	88
16	2617	307	36	163	563	56	3160	898	76	647	7	96	14	127
17	5089	859	37	2903	236	57	5144	516	77	3709	577	97	3385	659
18	1283	630	38	629	697	58	1657	298	78	2239	131	98	4685	119
19	1158	280	39	616	344	59	864	203	79	1658	517	99	473	408
20	5160	336	40	2696	518	60	3289	218	80	4310	502	100	653	202

Note: X,Y are the coordinates of positions; No. is referred by the P and D columns in Table A.3.

Table A.3. Origin, destination and loads of each task (Datasets 2 and 3)

No.	P	D	L	No.	P	D	L	No.	P	D	L	No.	P	D	L	No.	P	D	L

1	20	49	2	21	49	81	1	41	50	58	2	61	74	33	2	81	37	53	1
2	47	65	1	22	25	24	1	42	74	60	2	62	1	19	3	82	7	85	3
3	50	32	1	23	71	91	1	43	63	40	1	63	42	79	1	83	63	85	3
4	56	28	2	24	82	20	2	44	94	8	1	64	37	38	1	84	43	55	3
5	51	55	1	25	32	66	2	45	53	73	1	65	80	37	3	85	41	61	2
6	27	48	1	26	84	78	1	46	64	34	1	66	11	74	2	86	41	98	2
7	5	86	3	27	49	81	3	47	38	93	1	67	56	12	2	87	95	8	3
8	38	9	2	28	49	23	3	48	55	55	2	68	79	14	3	88	6	42	3
9	77	64	1	29	28	35	3	49	96	96	3	69	82	35	3	89	17	77	3
10	40	10	1	30	51	89	3	50	57	40	1	70	22	26	3	90	93	84	3
11	98	93	2	31	56	52	1	51	51	4	1	71	40	79	2	91	20	1	1
12	1	10	1	32	31	69	1	52	50	5	3	72	79	22	3	92	20	24	3
13	97	6	1	33	16	65	2	53	83	81	1	73	63	100	2	93	46	89	3
14	100	89	3	34	1	11	1	54	100	31	3	74	46	74	3	94	57	69	1
15	58	66	1	35	4	98	1	55	85	49	1	75	31	19	3	95	52	81	2
16	77	91	2	36	63	46	2	56	32	84	1	76	41	27	1	96	50	31	3
17	37	96	2	37	51	82	3	57	82	51	3	77	57	71	2	97	90	64	1
18	53	52	1	38	66	80	1	58	73	53	3	78	48	43	3	98	48	10	2
19	76	58	1	39	63	78	1	59	83	40	3	79	48	99	3	99	40	86	1
20	94	30	1	40	27	24	2	60	44	38	1	80	53	13	2	100	20	97	1

Note: P, D are the origin and destination positions (referring to the column "No." in Table A.2) of a task; L is the number of loads of a task.

Table A.4. Available position and time of each vehicle (Datasets 2 and 3)

No.	U_v	R_v /minute	No.	U_v	R_v /minute
1	59	8	6	22	2
2	77	9	7	99	6
3	89	2	8	19	9
4	29	6	9	16	0
5	57	8	10	64	1

Note: The values of the column "U_v" refer to the column "No." in Table A.2.

References

Chen, P., Guo, Y., Lim, A., Rodrigues, B., 2006. Multiple crossdocks with inventory and time windows. *Computers & Operations Research* 33(1), 43-63.

Cortes, C.E., Matamala, M., Contardo, C., 2010. The pickup and delivery problem with transfers: formulation and a branch-and-cut solution method. *European Journal of Operational Research* 200(3), 711-724.

Dao, S.D., Abhary, K., Marian, R., 2017. A bibliometric analysis of Genetic Algorithms throughout the history. *Computers & Industrial Engineering* 110(1), 395-403.

Derigs, U., Pullmann, M., 2014. Solving multitrip vehicle routing under order incompatibilities: A VRP arising in supply chain management. *Networks* 64(1), 29-39.

Derigs, U., Pullmann, M., Vogel, U., 2013. Truck and trailer routing - Problems, heuristics and computational experience. *Computers and Operations Research* 40(2), 536-546.

Dohn, A., Rasmussen, M.S., Larsen, J., 2011. The vehicle routing problem with time windows and temporal dependencies. *Networks* 58(4), 273-289.

Drexl, M., 2012. Synchronization in vehicle routing - A survey of VRPs with multiple synchronization constraints. *Transportation Science* 46(3), 297-316.

Drexl, M., 2013. Applications of the vehicle routing problem with trailers and transshipments. *European Journal of Operational Research* 227(2), 275-283.

Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley.

Gschwind, T., 2015. A comparison of column-generation approaches to the Synchronized Pickup and Delivery Problem. *European Journal of Operational Research* 247(1), 60-71.

Ioachim, I., Desrosiers, J., Soumis, F., Bélanger, N., 1999. Fleet assignment and routing with schedule synchronization constraints. *European Journal of Operational Research* 119(1), 75-90.

Joo, C.M., Kim, B.S., 2014. Block transportation scheduling under delivery restriction in shipyard using meta-heuristic algorithms. *Expert Systems with Applications* 41(6), 2851-2858.

Karakatič, S., Podgorelec, V., 2015. A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing* 27(1), 519-532.

Kim, G., Ong, Y.S., Heng, C.K., Tan, P.S., Zhang, N.A., 2015. City Vehicle Routing Problem (City VRP): A Review. *IEEE Transactions on Intelligent Transportation Systems* 16(4), 1654-1666.

Lin, S.W., Yu, V.F., Lu, C.C., 2011. A simulated annealing heuristic for the truck and trailer routing problem with time windows. *Expert Systems with Applications* 38(12), 15244-15252.

Marković, N., Drobnjak, T., Schonfeld, P., 2014. Dispatching trucks for drayage operations. *Transportation Research Part E: Logistics and Transportation Review* 70(1), 99-111.

Montoya-Torres, J.R., López Franco, J., Nieto Isaza, S., Felizzola Jiménez, H., Herazo-Padilla, N., 2015. A literature review on the vehicle routing problem with multiple depots. *Computers and Industrial Engineering* 79, 115-129.

Morais, V.W.C., Mateus, G.R., Noronha, T.F., 2014. Iterated local search heuristics for the Vehicle Routing Problem with Cross-Docking. *Expert Systems with Applications* 41(16), 7495-7506.

Park, C., Seo, J., 2012. A GRASP approach to transporter scheduling and routing at a shipyard. *Computers and Industrial Engineering* 63(2), 390-399.

Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L., 2013. A review of dynamic vehicle routing problems. *European Journal of Operational Research* 225(1), 1-11.

Rousseau, L.-M., Gendreau, M., Pesant, G., 2013. The synchronized dynamic vehicle dispatching problem. *INFOR* 51(2), 76-83.

Salazar-Aguilar, M.A., Langevin, A., Laporte, G., 2012. Synchronized arc routing for snow plowing operations. *Computers & Operations Research* 39(7), 1432-1144.

Salazar-Aguilar, M.A., Langevin, A., Laporte, G., 2013. The synchronized arc and node routing problem: Application to road marking. *Computers & Operations Research* 40(7), 1708-1715.

Stern, H., Chassidim, Y., Zofi, M., 2006. Multiagent visual area coverage using a new genetic algorithm selection scheme. *European Journal of Operational Research* 175(3), 1890-1907.

Syswerda, G., 1990. *Schedule Optimization Using Genetic Algorithms*. Van Nostran Reinhold, New York.

Villegas, J.G., Prins, C., Prodhon, C., Medaglia, A.L., Velasco, N., 2013. A matheuristic for the truck and trailer routing problem. *European Journal of Operational Research* 230(2), 231-244.

Wen, M., Larsen, J., Cordeau, J.-F., Laporte, G., 2009. Vehicle routing with cross-docking. *Journal of the*

Operational Research Society 60(1), 1708-1718.

Xue, Z., Zhang, C., Lin, W.H., Miao, L., Yang, P., 2014. A tabu search heuristic for the local container drayage problem under a new operation mode. *Transportation Research Part E: Logistics and Transportation Review* 62, 136-150.

Zhang, R., Lu, J.C., Wang, D., 2014. Container drayage problem with flexible orders and its near real-time solution strategies. *Transportation Research Part E: Logistics and Transportation Review* 61, 235-251.

Highlights

- Synchronize multiple unit-load vehicles for transporting cargos of various sizes.
- A sequential insert algorithm is developed based on practical experiences.
- A fast greedy insert algorithm is developed for large-scale instances.
- A genetic algorithm is developed based on the sequential insert algorithm.
- The synchronization parameters and algorithm performances are analyzed.