applied sciences

MDPI

Article

A Decomposition Algorithm for Dynamic Car Sequencing Problems with Buffers

Haida Zhang and Wensi Ding *

School of Mechanical and Automotive Engineering, South China University of Technology, Guangzhou 510640, China; zhdhnn@163.com
* Correspondence: dwszhd2023@163.com

Abstract: In this paper, we research the dynamic car sequencing problem with car body buffer (DCSPwB) in automotive mixed-flow assembly. The objective is to reorder the sequence of cars in the paint shop using the post-painted body buffers to minimize the violation of constraint rules and the time cost of sequencing in the general assembly shop. We establish a mathematical model of DCSPwB and propose a decomposition-based algorithm based on the dynamic genetic algorithm (DGA) and greedy algorithm for delayed car release (PGDA). Experiments are conducted based on production orders from actual companies, and the results are compared with the solution results of the underlying genetic algorithm (GA) and greedy algorithm (GDA) to verify the effectiveness of the algorithm. In addition, the effect of buffer capacity on DCSPwB is investigated.

Keywords: mixed-flow assembly; car sorting; decomposition algorithm; genetic algorithm; greedy algorithm

Citation: Zhang, H.; Ding, W. A Decomposition Algorithm for Dynamic Car Sequencing Problems with Buffers. *Appl. Sci.* **2023**, *13*, 7336. https://doi.org/10.3390/app13127336

Academic Editor: Dimitris Mourtzis

Received: 11 May 2023
Revised: 14 June 2023
Accepted: 19 June 2023
Published: 20 June 2023

1. Introduction

The introduction of Henry Ford's Model T revolutionized automobile manufacturing, and its creative assembly line has greatly reduced the cost of automobile production. As the automotive industry continues to evolve, product requirements and production system requirements have changed dramatically, and orders with "small lot size, multiple varieties, and customization" have made the mixed-flow production model increasingly popular. The so-called mixed-flow production model means that different body types, different colors, and different parts are produced on the same production line, making the production process more complex. Assembly line balancing and car sequencing are the two main issues for effective operation of mixed-flow assembly lines [1]. The assembly line balancing problem has been widely studied since its introduction [2–6]; scholars have constructed different models for the simple assembly line balancing problem (SALBP) and the generalized assembly line balancing problem (GALBP) by considering the number of workstations, productivity, assembly line cost, smoothness index of parts consumption, and workstation idle time to optimize the solution.

In addition to the assembly line balancing problem, the car sequencing problem (CSP) of automobiles is equally important. There are usually multiple departments in automobile production, such as welding shop, painting shop, and final assembly shop, etc. Each shop has different preferences for the sequence of car production. For example, the painting shop tends to limit the car production sequence by car color in order to save the cost of cleaning paint nozzles [7], and the final assembly shop tends to limit the production by model and configuration based on the consideration of man hours and parts consumption. Since the constraints of each shop on the mixed-flow assembly line are different, each shop has different car sequencing goals, which leads to production scheduling that cannot follow the same sequence of continuous production. In particular, the sequence difference between the paint shop and final assembly shop is large. For this situation, there are two solutions. The first one is to generate a fixed compromise sequence by considering the production

needs of multiple shops simultaneously. Cordeau et al. [8], Gagné et al. [9], Mansouri [10], Prandtstetter and Raidl [11], and McMullen [12] have described how to generate the initial production sequence, and it is obvious that the generated such sequences all violate the constraint rules of each workshop to varying degrees. Another approach is to create a buffer between these two shops to reorder the cars to obtain the desired sequence, at which point the CSP evolves into the car sequencing problem with buffer (CSPwB). There are various forms of buffer zones, and Boysen [13] and Wortmann [14] summarized several structural forms of buffer zones:

- Pull-out buffer: Cars at any position in the car sequence can be removed and placed in the backward buffer, and then inserted into the appropriate position as the car sequence advances, so that cars can be moved to any position later in the sequence, and the advancement of the car sequence is limited by the number of cars allowed out of the column.
- Ring buffer: Multiple cars can be selected from the car sequence into the ring buffer, and cars from any position in the ring buffer can be put back into the sequence.
- Automated storage and retrieval system (AS/RS): Consisting of hundreds of buffer points, each buffer point can be accessed individually to generate the desired sequence of cars, and the flexibility of car sequencing is influenced by the number of buffer point locations.
- Parallel buffer: It consists of multiple parallel lanes (Figure 1), where cars from the upstream workshop are put into the parallel lanes, and then the first car of a lane is selected to join the downstream car production sequence according to the needs of the downstream workshop, and cars can only move in one direction in the lane, following the "FIFO" principle.

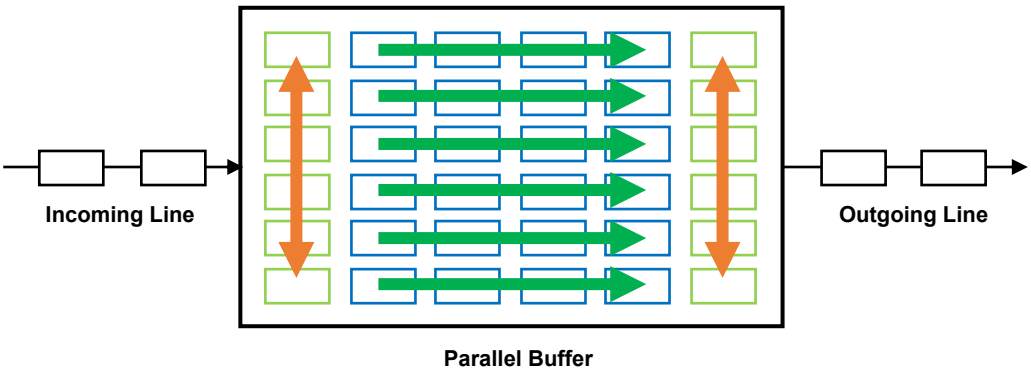

Figure 1. Parallel buffer.

Parallel line buffer is a more used class of buffer in automotive companies, and this paper also deals with solving CSP for parallel line buffer. We propose a decompositional algorithm to solve DCSPwB and reorder the car sequences in the paint shop using the post-painted car body buffer to find the optimal car sequences that satisfy all the hard constraints of the assembly shop as well as the appropriate trade-offs of various soft constraints, minimizing the violation of the constraint rules and the time cost of sequencing for the assembly shop. The whole method is divided into two parts: a heuristic inbound method based on simple rules and an outbound method based on a dynamic genetic algorithm and a greedy algorithm for delaying car outbound. We consider the time cost of the car sorting process as a soft constraint.

The rest of this paper is organized as follows. A literature review is presented in Section 2. Section 3 provides a detailed description of DCSPwB and establishes a mathematical model. In Section 4, a decomposed algorithm is proposed to divide the problem into two phases: inbound and outbound. A simple rule-based heuristic inbound algorithm, a greedy algorithm based on the delayed car outbound method, and a dynamic-genetic-algorithm-based car outbound method are described. The numerical experiments in Section 5 verify

the effectiveness of the proposed algorithms and investigate the effect of buffer capacity on the flexibility of car sorting. Section 6 describes the conclusions of this research.

2. Literature Review

CSP exists in a large number of automotive production companies and has gradually developed into a classical type of planning and scheduling problem. CSP was first proposed by Parrello et al. [15] and gradually gained attention. Kis et al. [16] proved that CSP is an NP-hard problem. Solnon et al. [17] divided the methods for solving CSP into two categories: exact methods (constraint planning, integer planning, etc.) and heuristic methods (genetic algorithms, ant colony optimization algorithms, etc.). Boysen et al. [1] classified three types of sorting methods in mixed-flow assembly based on different sorting objectives: mixed model sorting, car sorting, and level scheduling. Drexl and Kimms [18] proposed an integer planning model for CSP. Thiruvady et al. [19] solved CSP based on mixed integer programming (MIP) with large neighborhood search (LNS) by minimizing the violation of car separation rules. Estellon et al. [20] proposed two local search methods that use invariants to speed up the search for the optimal solution of the algorithm to solve the CSP. Prandtstetter and Raidl [11] proposed two methods to solve CSP, i.e., integer linear programming (ILP) and hybrid variable neighborhood search, and showed the superiority of the algorithms by experimental examples. Gavranović [21] solved the CSP with color by improving the variable neighborhood search (VNS) results with the Tabu metaheuristic. Siala et al. [22] studied the effect of different heuristics on CSP and proposed a slack-based filtering algorithm.

For CSPwB, Bulgak [23] developed an artificial-neural-network-based metamodel simulation combined with the genetic algorithm (ANN-GA) to search for optimal solutions. Muhl et al. [24] investigated the effect of using different metaheuristics on solving CSPwB. Spieckermann et al. [25] used a branch-and-bound algorithm to solve CSPwB with the objective of minimizing the number of color switches in the paint shop. Yu et al. [26] proposed a new algorithm of gravity-like mechanism to achieve the sequencing of cars in the buffers by encoding them as particles. Moon et al. [27] proposed a store/retrieve algorithm and simulated the proposed algorithm to verify the effectiveness of the proposed algorithm. Pereira et al. [28] developed a new branch-and-bound algorithm that exploits the symmetry of the problem and improves the efficiency of the algorithm.

Currently, these studies on CSP and CSPwB tend to consider only the degree of violation of the rules and the operational efficiency of the algorithm, and do not consider the time cost of car sorting. However, in the actual enterprise production, sometimes, there is a surge of orders and the production beat needs to be accelerated, and the excessive car sequencing time will affect the production schedule. Furthermore, previous studies were often based on static reordering [13], i.e., all information is known before car sequencing, and thus, decomposing the problem into a series of static problems within rolling planning. The algorithm proposed in this paper is based on the dynamic sequencing of cars, i.e., the sequence of cars before reaching the buffer is not known, which makes the car sequencing problem an online problem.

3. Problem Description and Mathematical Model

In this section, we develop a detailed mathematical model for DCSPwB. Consider an initial sequence of N cars of different models: $i = 1, 2, \ldots, N$, O is the set of options required for different models of cars, and for any option in the set, the capacity constraint, i.e., $r_o : s_o$, is defined, and the number of cars with option o cannot exceed r_o in the sequence s_o of cars requiring option o. The capacity constraints are determined based on the production characteristics of the assembly plant downstream of the buffer. Violation of these constraints can cause problems, such as excessive workload of assembly plant workers and uneven consumption of parts. The following is a simple example to illustrate the capacity constraint, as shown in Table 1, which considers a car production schedule with four types, i.e., A, B, C, and D, for a total of five cars. Different types of cars need to

satisfy different options; for example, Class A cars need to satisfy options o_1 and o_2, while Class B cars only need to satisfy o_1. The constraints associated with o_1 and o_2 are $1/2$ and $1/3$, respectively. These five cars can generate $5! = 120$ kinds of car sequences; here, we use the "sliding window" method to count the number of violations. This method will be repeated to amplify the number of violations of the option constraints, which is equivalent to increasing the weight of the "problem car", so that the algorithm tends to disperse the "problem car". For example, if the sequence of cars after reordering in the buffer is [B, A, A, C, D], then the subsequence [B, A], [A, A] violates the constraint o_1, and the subsequence [A, C, D] violates the constraint o_2, and thus, the total number of violations of the sequence is three. Here, the second car is a "problem car", and removing or replacing the second car will reduce the number of violations by two.

Table 1. A simple example of a car sequencing problem.

Car Type	A	B	C	D	
o_1	✓	✓			1/2
o_2	✓		✓	✓	2/3
Amount	2	1	1	1	

The buffer consists of L parallel lanes, each of which can hold up to V cars. The initial sequence of cars is reordered through the buffer to generate a new sequence into the downstream shop, and only one car can enter and leave the buffer at a time, so the flexibility of reordering is related to the buffer capacity. We also consider the time cost of DCSPwB and add it to the objective function. In addition, the following assumptions are made:

1. Each car takes the same amount of time to move one space in either lane of the buffer and can only move in one direction.
2. The buffer continues to function normally, regardless of fault conditions.

According to the above expression and the definition of symbols in Table 2, DCSPwB is defined as follows:
Min:

$$F = \sum_{o \epsilon O} w_o \cdot \sum_{j=1}^{\min(j+s_o-1,N)} z_{oj} + w_t \cdot sc_t \sum_{i=1}^{N} \sum_{l=1}^{L} y_{il}[(V-1) \cdot TL + te_l + tf_l] \quad (1)$$

Subject to:

$$\sum_{l=1}^{L} y_{il} = 1, \forall i = 1, \ldots, N \quad (2)$$

$$\sum_{i=1}^{N} y_{il} \leqslant V, \forall l = 1, \ldots, L \quad (3)$$

$$\sum_{i=1}^{N} c_{il} = 1, \forall j = 1, \ldots, N \quad (4)$$

$$\sum_{k=1}^{N} c_{ik} \cdot k - \sum_{k=1}^{N} c_{jk} \cdot k \leqslant MI\left(2 - y_{il} - y_{jl}\right),$$
$$\forall i = 1, \ldots, N-1, j = i+1, \ldots, N, l = 1, \ldots, L \quad (5)$$

$$\sum_{i=1}^{N} \sum_{\tau=j}^{j+s_0-1} a_{io} \cdot c_{i\tau} \leqslant r_0 + M \cdot z_{oj},$$
$$\forall o \epsilon O, j = 1, \ldots, N - s_o + 1 \quad (6)$$

Table 2. Variable and parameter definitions.

N	Total number of cars, index i.
O	Collection of options, index o.
L	Number of lanes in the buffer, index l.
V	Capacity per lane.
w_o	Option weights.
s_o	A continuous sequence of cars, some of which require the option o.
r_o	The maximum number of cars with option o allowed in a continuous sequence of cars s_o.
$s_o : r_o$	Capacity constraint, i.e., the maximum number of cars with option o allowed in a continuous sequence of cars s_o.
z_{oj}	Binary variable: 1, if the car sequence s_o starting from position j satisfies the constraint $s_o : r_o$, 0, otherwise.
M	Scaling for $(s_o : r_o)$.
y_{il}	Binary variables: 1, if car i is in lane l, 0, otherwise.
c_{ij}	Binary variables: 1, if car i is in position j of the sequence, 0, otherwise.
MI	Large integers.
a_{io}	Binary variables: 1, if car i needs option o, 0, otherwise.
TL	The time it takes for a car to move one space in the buffer lane.
t_{el}	Time it takes for a car to enter the buffer lane l.
t_{fl}	Time it takes for a car to exit from the buffer lane l.

The objective Equation (1) is to minimize the number of violations of the car sequence in all options and the time cost of car sorting, balancing the importance of time by the weight coefficient w_t. sc_t is the time scale scaling factor, because the time cost of car sorting is not consistent with the scale of the number of violations and the degree of variation is different, and the scaling of sorting time is needed. Constraint Equation (2) ensures that each car can only be assigned to one lane in the buffer. Constraint Equation (3) ensures that the number of cars in each lane of the buffer does not exceed its capacity. Constraint Equation (4) guarantees that each car can only appear once in the car sequence. Constraint Equation (5) guarantees that for any two cars i and j, if they are assigned to the same lane, the position of car i in the lane of the buffer, and in the sequence of cars after reordering, is before car j. Constraint Equation (6) checks whether rule violations occur, while allowing these rules to be violated when there is no feasible solution. Constraints Equations (2)–(5) are hard constraints, i.e., constraints that need to be fully satisfied, and constraint Equation (6) is a soft constraint, i.e., a constraint that is allowed to be violated at some cost. Therefore, the essence of the optimization process for this problem is to satisfy the soft constraints at the least costly expense while satisfying all hard constraints.

The next section gives our proposed decomposition approach to solve the problem, which divides the problem into two phases to solve: the inbound phase (cars enter the buffer) and the outbound phase (cars exit the buffer), for which different heuristics are applied.

4. DCSPwB Decompositional Algorithm

Since the whole sorting process is dynamic and is a continuous input–output process, the sequence of cars after reordering is subject to both the constraints of the paint shop and

the rules of the final assembly shop, and both car inbound and outbound affect the number of rule violations as well as the time cost of car sequencing. In dynamic sequencing, since the sequence of cars before arriving at the buffer is not known, we use a simple rule-based heuristic algorithm to fill the buffer in the car-in phase; in the car-out phase, we propose two different algorithms: a greedy algorithm to delay the car-out (PGDA), and a dynamic genetic algorithm (DGA) to release the cars from the active sequence in the buffer to the final assembly plant. The so-called active sequence is the sequence consisting of the cars in the first position of all lanes [1].

4.1. Simple Rule-Based Heuristic Entry Algorithm

In the car entry phase, we introduce a scoring model, where the level of the score represents the magnitude of the match, and the scoring model is as in Equation (7). Where $f_i(l)$ denotes the matching degree of car i with lane l, n denotes the number of rules, r denotes the priority of the rules, I is a positive integer determined by the number of rules, and the rules are formulated based on the attributes of the car. For example, if the configuration of a car is 4WD and hybrid, then the attributes of the car are {4WD, hybrid}. Here, we only care about the attributes related to the constraints of the final assembly plant, such as the driving mode and power source of the car, etc. The attributes related to the paint shop, such as the color of the car, are not relevant to the optimization objective Equation (1) and are not considered here.

$$\underset{l=1,\ldots,L}{Max}\ f_i(l) = \begin{cases} \sum_{r=1}^{n} [u(r) \cdot 2^{n-r}], & \text{if lane } l \text{ is not empty} \\ \sum_{r=1}^{n} 2^{n-r} - I, & \text{if lane } l \text{ is empty} \\ 0, & \text{if } l = x \text{ and } arg\ Max\ f_{i-1}(l) = x \end{cases} \tag{7}$$

$$u(r) = \begin{cases} 1, & \text{if } d_{lr} = car_{ir} \\ 0, & \text{if } d_{lr} \neq car_{ir} \end{cases} \tag{8}$$

where $u(r)$ is the attribute consistency metric and d_{lr} and car_{ir} denote the attributes of lane l and car i, respectively, when based on rule r. Once a car enters in a lane, the attributes of the lane remain consistent with that car. We also consider the time problem when cars enter the buffers. Different lanes have different distances from the entrance of the buffers; if a certain lane is close to the entrance of the buffers, the time for a car to enter that lane is shorter than the time for a car to move one space in that lane, which will result in a car waiting situation and increase the time cost of the car sorting process, so we avoid this situation by setting the matching degree of that lane to 0. According to the matching degree scoring model, the inbound heuristic rules in order of priority from highest to lowest are:

1. Select the lane with the best match.
2. Select the lane with the largest remaining capacity.
3. Select the lane that takes the shortest amount of time to enter the buffers.

A simple rule-based heuristic entry is shown in Algorithm 1. Algorithm 1 uses the car i that needs to enter the buffers and the set T of the time it takes for the car to enter each lane of the buffers as input. In line 1, the highest match $maxScore$ of all lanes with car i is initialized to 0. In line 2, the set BL consisting of the lanes with the highest matching degree is initialized to empty. The loop in rows 3–7 updates the highest matching $maxScore$ of all lanes with car i. The loop in lines 8–12 updates the set BL consisting of the lanes with the highest matching degree. If the lane with the highest matching degree is not unique, in line 14, the function $choic\_margin(BL)$ is used to calculate the lane with the largest remaining capacity and update BL. If the lane with the largest remaining capacity is not unique, the lane that takes the shortest time for car i to enter the buffer is calculated by the function $choic\_time(BL, T)$ and BL is updated, the lane assigned to car i is returned at the end of the algorithm. The time complexity of this algorithm is $O(L)$. The algorithm matches different lanes for cars with different attributes based on the information of cars that are about to be warehoused and the information of the buffers, and does not depend

on the information of car sequences in the upstream workshops to meet the dynamic sequencing requirements of cars.

Algorithm 1: Heuristic storage algorithm for simple rules

Input: A car i, Set of time required for car to enter each lane of buffer T.
Output: The set of lanes with the highest matching degree BL.
$maxScore \longleftarrow 0$
$BL \longleftarrow \emptyset$
for $l = 1, \ldots, L$ **do**
  if $f_i(l) > maxScore$ **then**
    $maxScore \longleftarrow f_i(l)$
  end
end
for $l = 1, \ldots, L$ **do**
  if $f_i(l) = maxScore$ **then**
    append l to BL
  end
end
if $length(BL) > 1$ **then**
  $BL \longleftarrow choic_margin(BL)$
  if $length(BL) > 1$ **then**
    $BL \longleftarrow choic_time(B, T)$
  end
end
return BL

4.2. Dynamic Genetic Algorithm

Genetic algorithms (GA) were first proposed by Holland and have been shown to be used to solve different types of optimization problems [29–32]. In this paper, a novel dynamic genetic algorithm (DGA) is proposed to solve the vehicle outgoing problem by improving the crossover and variation operators of the GA to dynamically adjust the crossover and variation probabilities during the iteration process, and thus, adjust the search range of the algorithm. This section is organized according to the main steps of the genetic algorithm: (1) the way of encoding the solution and population initialization, (2) selection, (3) dynamic crossover algorithm, and (4) dynamic mutation algorithm.

4.2.1. Encoding Method of Solution with Population Initialization

Before using the genetic algorithm, the encoding of the solution is first determined. The encoding of the solution proposed in this paper is shown in Figure 2. The encoding shown here is for the active sequence of a buffers with six parallel lanes. The position of the gene represents the order in which the cars exit the buffer, and the value of the gene indicates the lane number in which the car exiting the buffer is located. For example, the first gene of the chromosome has a characteristic value of 5, indicating that the car in the first position of lane 5 exits the buffer first, followed by the car in the first position of lane 3. This encoding ensures that the cars in the active sequence of the buffer are exited sequentially.

As described in Section 3, each car has a different option o and needs to satisfy different capability constraints $s_o : r_o$, so the encoding of the solution also needs to be able to represent the configuration of the car. We use a multi-layer code to indicate the configuration information such as the driving method and power source of the car. As in Figure 3, it can be indicated that the car in the first position of lane 5 has drive mode 1 and power source 2. For population initialization, we generate the initial population using a prior knowledge-based guided population initialization method. From the encoding of the solution, it is known that the eigenvalues of each gene of the chromosome are mutually

exclusive, and based on this prior knowledge, the individuals that meet the requirements are selected among the randomly generated population individuals to ensure the feasibility of the initial population. Using the population initialization method guided by prior knowledge will speed up the convergence of the algorithm as well as improve the stability of the algorithm [33].

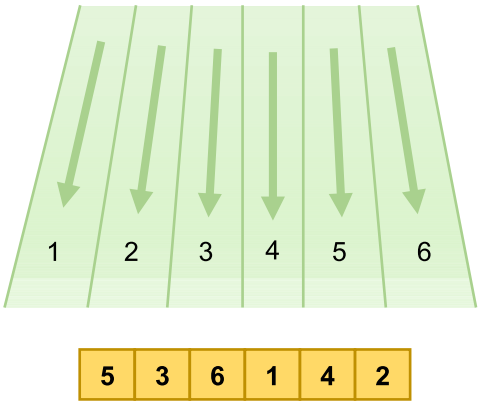

Figure 2. The way the solution is encoded.

5	3	6	1	4	2
1	2	1	2	2	1
2	1	2	1	2	2
…	…	…	…	…	…

Figure 3. Coding method for solutions with car configuration.

4.2.2. Selection

Selection is the process of selecting the best individuals from an old population to form a new population in order to reproduce the next generation of individuals. Here, we use binary tournament selection (TS), which is based on the idea that two individuals are randomly selected from the population at a time, and the individual with the highest fitness is chosen to enter the next generation population. TS has been widely used because of its low time complexity ($O(n)$) and the fact that it can be parallelized. Algorithm 2 shows the general process of TS. It can be seen that the selection is based on the fitness of individuals, so we need to define the fitness function. Each chromosome represents a set of car outgoing sequences, and adding different outgoing sequences to the already outgoing car sequences will increase the number of violations and time cost differently, i.e., cause different degrees of increase in the objective function Equation (1). We use the inverse of the amount of change in the objective function Equation (1) caused by different chromosomes as the fitness function:

$$Fitness = \frac{1}{\triangle F} \tag{9}$$

where the larger the $\triangle F$ of an individual, the lower its fitness and the lower the probability that the individual will continue to the next generation.

Algorithm 2: GA TS

Input: Num of chromosomes in the population n, vector of chromosomes fitness values v, number of parental chromosomes m.

$parentIndex = randi(n, 1, m)$
$vParent = v(parentIndex)$
$[vParentSort, vParentIndex] = sort(vParent, "descend")$
$idx = vParentIndex(1)$
return idx

4.2.3. Dynamic Crossover Algorithm

The crossover operator in GA replaces and recombines part of the structure of two parent chromosomes to generate a new chromosome. The new chromosome retains some of the original features of the parent chromosome and generates new features, thus improving the search ability of the genetic algorithm. In the traditional genetic algorithm, the crossover probability is fixed during the iteration; we propose a dynamic crossover algorithm that dynamically adjusts the crossover probability according to the number of iterations and the fitness of the population. The crossover probability is calculated as in Equations (10) and (11):

$$P_c^g = p_c^{max} - \frac{(g-1)^\gamma \cdot \left(p_c^{max} - p_c^{min}\right)}{\left(g^{max} - 1\right)^\gamma} \tag{10}$$

$$\gamma = k_c \cdot \left(Fitness_g^{max} - Fitness_g^{avg}\right) \tag{11}$$

where p_c^g denotes the crossover probability of the gth generation, p_c^{max} denotes the maximum crossover probability, p_c^{min} denotes the minimum crossover probability, g^{max} denotes the maximum number of iterations, $Fitness_g^{max}$ denotes the maximum fitness value of the gth generation population, $Fitness_g^{avg}$ denotes the average of the fitness values of the gth generation population, and k_c is a constant coefficient. As the number of iterations increases, the crossover probability decreases continuously, which reduces the abrupt changes of unsuitability in the population and ensures the convergence ability of the algorithm. Meanwhile, the speed of crossover probability reduction is adjusted by the difference between the maximum value of population fitness and the mean value of fitness to ensure the global optimization-seeking ability of the algorithm.

For the crossover method, we use the PMX crossover operator [34], a crossover operator that is guaranteed to produce valid chromosomes. PMX determines the crossover region by randomly selecting two crossover points, and the crossover region is populated directly to the offspring chromosome, for genes outside the crossover region in the offspring, which needs to be determined based on the mapping relationship established by the crossover region. Figure 4 shows an example of the PMX crossover operator, using the crossover region of Parent 2 to populate Offspring 1, and for genes outside the crossover region of the Offspring 1 chromosome, Parent 1 is used to populate. Finally, if there is an overlap with a gene in the crossover region in Offspring 1, the mapping relationship is used. For Figure 4, the mapping relationships established through the crossover region are 5-4, 1-2, and 3-6, and these mapping relationships ensure the validity of the chromosomes generated after the crossover.

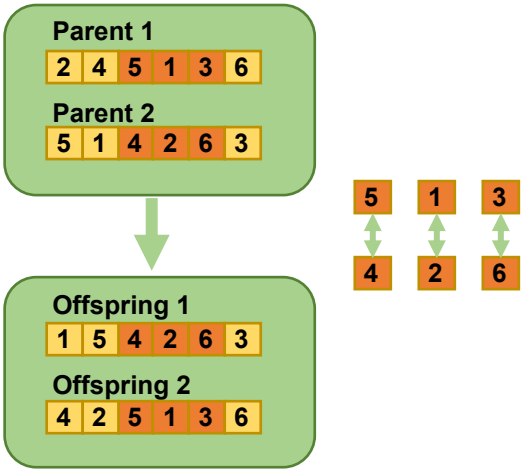

Figure 4. Partial match crossover (PMX).

4.2.4. Dynamic Mutation Algorithm

If there is only selection and crossover without mutation, it will not be possible to search in the space beyond the initial gene combination, making the algorithm fall into a local optimum solution and stop early in the iteration, so the mutation operator is necessary for the algorithm. The mutation operator in GA generates new chromosomes by changing some genes on the chromosome, which can improve the diversity of the population and reduce the risk of the algorithm falling into a local optimum solution. For the mutation probability, similar to the crossover probability, we propose a dynamic mutation algorithm that dynamically adjusts the mutation probability according to the number of iterations and the fitness of the population. The mutation probability is calculated as in Equations (12) and (13):

$$P_m^g = p_m^{max} - \frac{(g-1)^\gamma \cdot \left(p_m^{max} - p_m^{min}\right)}{\left(g^{max} - 1\right)^\gamma} \tag{12}$$

$$\gamma = k_m \cdot \left(Fitness_g^{max} - Fitness_g^{avg}\right) \tag{13}$$

where p_m^g denotes the mutation probability of the gth generation, p_m^{max} denotes the maximum mutation probability, p_m^{min} denotes the minimum mutation probability, and k_m is a constant coefficient. As the number of iterations increases, the mutation probability decreases continuously, while the rate of crossover probability reduction is adjusted by the difference between the maximum value of the population fitness and the mean value of the fitness, which ensures the convergence ability and global merit-seeking ability of the algorithm. In the mutation method, we use the swap mutation operator, which swaps two genes on the chromosome according to the mutation probability, as in Figure 5, and the resulting new chromosome is genetically identical to the original chromosome, which ensures the validity of the new chromosome generated after the mutation.

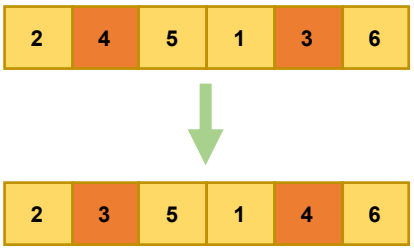

Figure 5. Exchange mutation.

4.3. Greedy Algorithm for Postponed Car Release

The greedy algorithm (GDA) is an algorithm that takes the best or optimal choice in the current state at each step of selection, and thus, hopes that the result is the best or optimal. GDA is not considered in terms of overall optimality, and the solution obtained each time is a local optimal solution based on some strategy [35,36]. GDA has been studied in solving static car sequencing problems [37], but its performance on dynamic car sequencing problems still needs to be further explored. Here, the GDA-based outbound method requires releasing the car in the active sequence of the buffer that causes the minimum sum of the number of new violations and the sorting time cost of the outbound car sequence. The number of new violations of the outgoing car sequence caused by the outgoing car is the sum of the number of violations for each option. For option o, if the outgoing car requires option o and the number of cars requiring option o in the outgoing car sequence s_o is equal to r_o, then the outgoing car causes one violation of option o. The number of additional violations caused by the outgoing car i is calculated according to Equation (14), where x denotes the position of the outgoing car in the sequence of already outgoing cars. The time cost of the outgoing car increase is calculated according to t_{fl}.

$$\begin{aligned} Vio(i) &= \sum_{o \in O} w_o \cdot \min\{a_{io}, z_{oq}\} \\ q &= \max\{0, x - s_o + 1\} \end{aligned} \tag{14}$$

Here, we improve the GDA and propose a greedy algorithm (PGDA) for delaying car exiting by setting the parameter M. Only when the number of cars in the buffer is larger than $(L \times V - M)$ are the cars allowed to exit the buffer, and the optimal exit of cars is solved by the greedy algorithm. The specific setting of parameter M needs to be obtained by simulation experiments.

5. Experiments and Results

We use two datasets to test the performance of the algorithm, based on data provided by a car manufacturer, and divide the dataset according to the total number of cars, $N \in \{60, 120, 180, 240, 300, 360\}$. Thus, a total of 12 sets of experimental arithmetic, denoted as I_60, II_60, I_120, II_120…, are obtained for the relevant configuration of the buffers, as described in more detail in Section 5.1. We compared the GDA-, PGDA-, GA-, and DGA-based methods for outbound cars. Section 5.2 shows the results of the numerical experiments.

5.1. Dataset

Table 3 shows some of the information of the dataset. For each dataset, we determine the order of the outgoing cars in the paint shop, i.e., the order of incoming cars in the buffers, the model information, and the options needed for different models, where each option is o_1 : 1/3, o_2 : 2/3, o_3 : 1/2. Parameters of the buffer are shown in Table 4. For the car sequencing problem in both datasets, violation of the constraint is unavoidable, i.e., Equation (1) $\sum_{o \in O} w_o \cdot \sum_{j=1}^{\min(j+s_o-1,N)} z_{oj} > 0$. We set different weights w_1, w_2, w_3 for the violation of different options. The weights represent the priority of different options and will be considered at different factories or at different times in the same factory. The value of the weights need to be determined according to the assembly capacity of the factory assembly plant, the order situation, etc. Here, the weights of options o_1, o_2, o_3 are w_1: 0.4, w_2: 0.3, and w_3: 0.2 and the weighting factors for time cost are w_t: 0.1 and sc_t: 0.01.

Table 3. Dataset introduction.

Order of Entering the Car	Car Type	o_1	o_2	o_3
1	K1		✓	
2	K2	✓		✓
3	K2	✓		✓
4	K1		✓	
...	...	...	...	...
22	K3		✓	
23	K1		✓	
...	...	...	...	...

Table 4. Buffer parameters.

Parameters	Value	Unit
L	6	
V	10	
TL	9	s
t_{el}	[18, 12, 6, 0, 12, 18]	s
t_{fl}	[18, 12, 6, 0, 12, 18]	s

5.2. Experimental and Computational Results

As described in Section 4.3, for PGDA, the value of the parameter M needs to be obtained by evaluating the quality of the solution for different experimental arithmetic cases. First the number of violations $VioP$ for the sequence of cars in the paint shop are counted before reordering, and the number of violations $VioM$ and the time cost tm of the sorting are obtained by the statistical algorithm when M takes different values on different test arithmetic cases, until the values of $VioM$ and tm values do not vary with M. The detailed results are shown in Figure 6, and it can be seen that when M = 2, $VioM$ tends to achieve the minimum value, tm is within an acceptable range, and the value of the objective function Equation (1) is optimal.

There is usually a range of values for each parameter of the genetic algorithm and different combinations of these parameter values greatly affect the performance of the algorithm. Here, the solution quality of the algorithm is evaluated by different experiments with different combinations of parameter values, and better values of the genetic algorithm parameters are obtained. In all experimental arithmetic cases using the genetic algorithm, the population size is 50, p_c^{max} is 1, p_c^{min} is 0.6, k_c is 3, p_m^{max} is 0.1, p_m^{min} is 0.01, k_m is 2, and the maximum number of iterations is 100. Based on the proposed simple rule-based heuristic entry algorithm and four different heuristic exit algorithms, i.e., GDA, PGDA, GA, and DGA, experiments were conducted on DCSPwB of different sizes, and the number of violations $VioP$ for the sequence of cars in the paint shop before reordering, $Vio\_GDA/Vio\_PGDA/Vio\_GA/Vio\_DGA$ for the sequence of cars after reordering by each algorithm, and the time cost $t\_GDA/t\_PGDA/t\_GA/t\_DGA$ were counted. The experimental results are shown in Table 5 and Figure 7. In Table 5, the first column identifies the test arithmetic, the next column shows the number of violations of the car sequence in the paint shop, and the next eight columns show the results of GDA, PGDA, GA, and DGA. Due to the random nature of the genetic algorithm, the results of the genetic algorithm are the average of five runs.

According to Table 5, we can see that PGDA, GA, and DGA all have a substantial reduction in the number of violations for the reordered car sequence, while GDA does not work well in solving the dynamic car sequencing problem. In Figure 7, we see the comparison between these four algorithms in different test cases. PGDA outperforms the other algorithms in terms of solution quality as well as time cost of sorting, and the objective function value F reaches the optimum. However, the choice of its parameter M often requires several experiments based on the buffer capacity and upstream workshop car sequences to evaluate the solution quality obtained with different parameters. The

solution quality of DGA and GA is comparable, but the convergence of DGA is better than GA. Figure 8 shows the convergence curves of several groups of genetic algorithms in the solution process. It can be seen that the DGA outperforms the GA in terms of convergence on both the population optimum and the mean value, and there is no risk of falling into a local optimum solution. Figure 7 also reflects that the algorithm performs differently on different datasets, and the sorting effect of the buffers is more influenced by the upstream shop, and the sequencing effect of the buffers will be reduced if the outgoing sequence of the paint shop differs significantly from the target sequence of the final assembly shop.

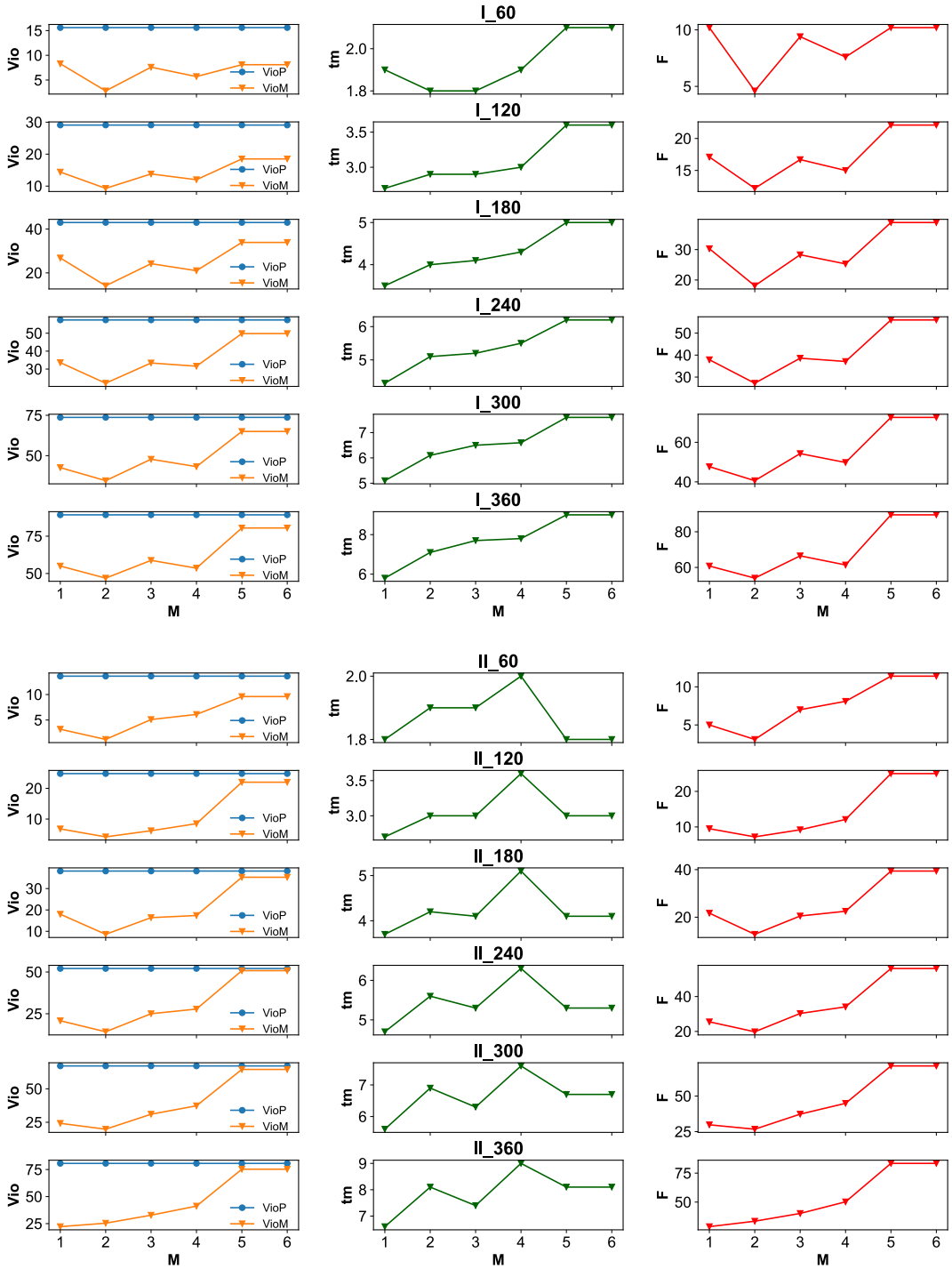

Figure 6. The quality of the PGDA-based solution when *M* takes different values.

Table 5. Calculation results of the four algorithms' examples.

Example	*VioP*	GDA		PGDA		GA		DGA	
		Vio_GDA	*t_GDA*	*Vio_PGDA*	*t_PGDA*	*Vio_GA*	*t_GA*	*Vio_DGA*	*t_DGA*
I_60	15.6	13.1	1.2	8.3	1.8	5.6	2.2	**5.1**	2.2
I_120	29.1	24.3	2.3	**9.3**	2.9	12.2	3.5	10.1	3.5
I_180	43	38.9	3.4	**14.1**	4.0	19.1	4.6	19.5	4.8
I_240	57.4	53.7	4.5	**22.1**	5.1	27.4	6.3	25.2	6.2
I_300	73.7	71.2	5.5	**34.5**	6.1	39.6	7.3	37.8	7.6
I_360	89.3	85.6	6.6	**46.9**	7.1	50.6	8.7	48.1	8.8
II_60	13.6	13	1.2	**1.2**	1.9	3.2	2.3	2.8	2.2
II_120	24.8	24.8	2.1	**4.2**	3.0	5.6	3.4	7.6	3.5
II_180	38.2	38.8	3.1	**8.6**	4.2	12.6	4.9	13.2	4.7
II_240	52.1	53.4	4.1	**14.2**	5.6	20.8	6.1	22.8	6.3
II_300	67.1	69.5	5.2	**19.8**	6.9	26.2	7.4	28.1	7.5
II_360	80.7	81	6.1	**25.4**	8.1	33.4	9.1	35.8	8.8

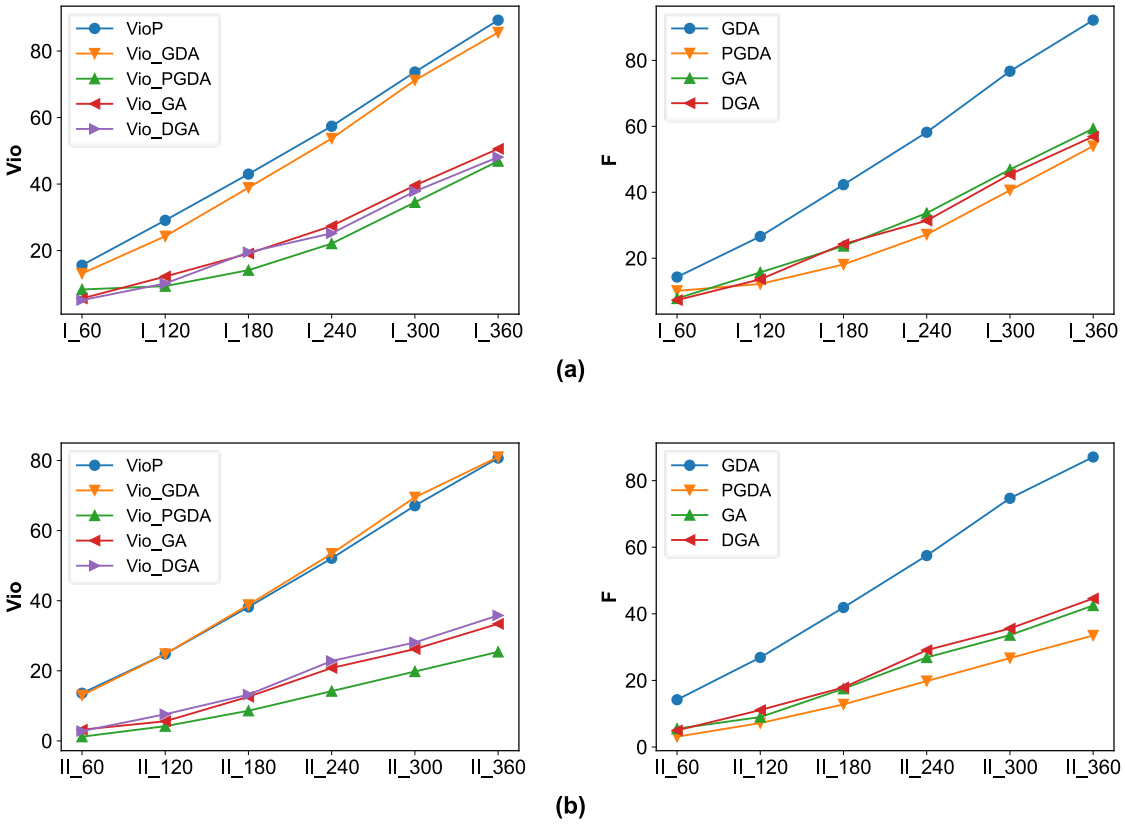

Figure 7. (**a**) Number of violations and objective function values after reordering of dataset I for each algorithm. (**b**) Number of violations and objective function values after reordering of dataset II for each algorithm.

5.3. The Effect of Buffer Capacity on Car Resequence

As the number of cars increases, the resequencing capability of the buffers is diminishing, as shown in Figure 9. It is not difficult to understand that since the capacity of the buffers is limited, their resequencing capability for cars is also limited. Here, we investigate the effect of the buffers capacity on its resequencing capability. Figure 10 shows the change of the average objective function value F on the test instances I_360 and II_360 using PGDA as the number of lanes L in the buffers increases. It can be seen that increasing the number of lanes improves the sequencing capability of the buffers, but this improvement in sequencing capability diminishes rapidly when the number of lanes in the buffer area reaches

a certain number. As the number of lanes continues to increase, the time cost of sorting increases, the objective function value F increases instead, and the sequencing ability of the buffer is weakened.

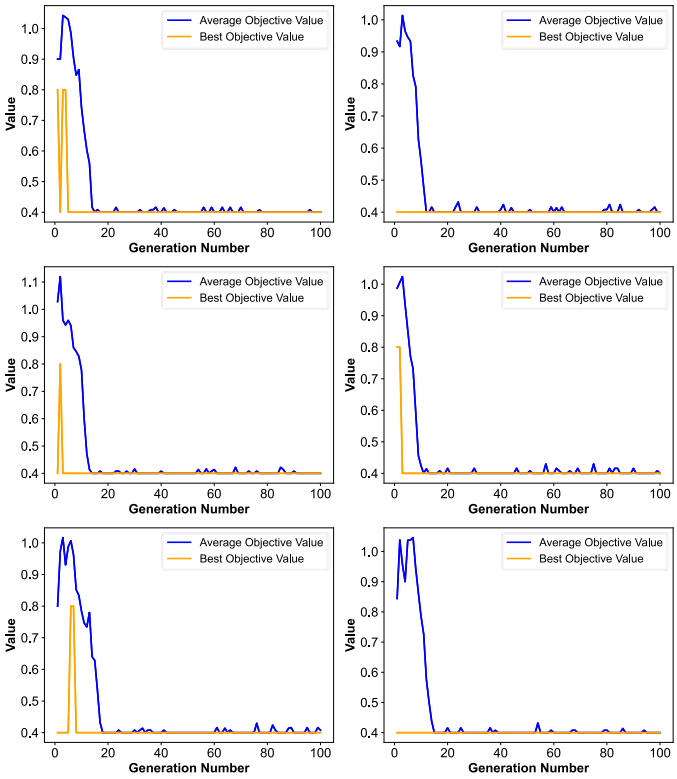

Figure 8. Convergence curves of GA (**left**) and DGA (**right**).

It can be seen that a certain number of lanes can already guarantee the sequencing capability of the buffer, and an excessive number of lanes will not only mismatch with the construction cost of the buffer, but even reduce the sequencing capability of the buffer.

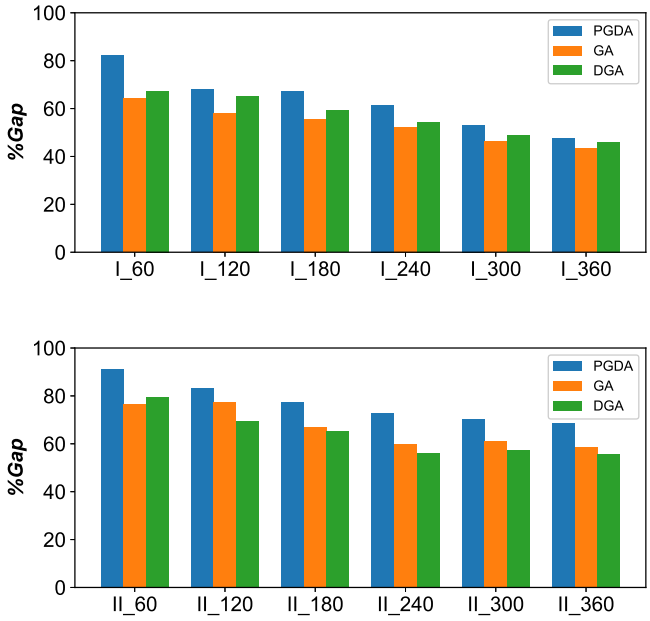

Figure 9. The percentage difference in the resequencing capacity of the buffer with different number of cars; $Gap(\%)$ is computed as $Gap(\%) = \frac{VioP - Vio\_PGDA}{VioP}$.

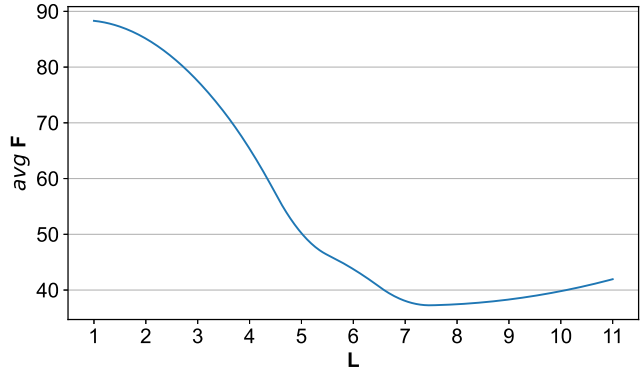

Figure 10. Effect of number of lanes on sequencing capability.

6. Conclusions

With the continuous development of the automotive industry, the mixed-flow production model is becoming more and more popular. In this paper, a decomposition-based algorithm is proposed to solve DCSPwB in automotive mixed-flow assembly, a mathematical model of DCSPwB is established, and the time cost of car sequencing is considered. Using the post-painted car body buffer, the car sequences in the paint shop are reordered to find the optimal car sequences that satisfy all hard constraints in the general assembly shop as well as properly trade-off various soft constraints, minimizing the violation of the constraint rules and the time cost of sequencing in the general assembly shop. The algorithm consists of two heuristic phases, a simple rule-based heuristic inbound phase and a PGDA-based as well as a DGA-based car outbound phase. In addition, we explore the effect of buffer capacity on DCSPwB.

To evaluate the performance of the proposed algorithms, 12 test examples are provided. The results show that car sequencing methods based on different exit algorithms are all effective in reducing the violation of downstream workshop constraint rules and that the time cost of car sequencing is acceptable. PGDA tends to achieve better results, but the determination of its parameter M needs to be obtained by evaluating the quality of different solutions. The quality of the solution of DGA is not as good as that of PGDA, but its parameters are easier to obtain. Meanwhile, our research shows that in DCSPwB, the sequencing ability of the buffers is weakened as the number of cars increases, and increasing the number of lanes improves the sequencing ability of the buffers, but this improvement is tiny and even reduces the sequencing ability of the buffers when the number of lanes in the buffers reaches a certain number, and a certain number of lanes is already able to guarantee the sequencing ability of the buffers.

Although the research in this paper is for a specific car sequencing problem, the solution is flexible enough to be extended to a wide range of reordering problems in reality. The general reordering rules are divided into hard and soft constraints, and modifying the weights of the different soft constraints while the determined compliance with the hard constraints can apply this method to other problems. In actuality, any problem with buffers and with resequencing nature can refer to this approach; for example, in the cross-docking distribution problem in supply chain logistics, how to plan the order of incoming and outgoing vehicles is the key to improve the efficiency of the distribution system. This is where cross-docking distribution becomes an online scheduling problem, with the warehouse serving as temporary storage with a buffers area. If different types of products unloaded are reordered within the warehouse, the scheduling difficulty of incoming and outgoing vehicles can be reduced. The object of resequencing here is different types of products, and the optimization goal can be the matching degree of products and shipping vehicles, while the effectiveness of the method can be tested on relevant types of instances in the future, such as cross-docking distribution problems in supply chain logistics. In this paper, the parameter M of PGDA is obtained by evaluating the quality of the solution on different test cases, and the possibility of determining the parameter M by

some other methods can be investigated in the future. In addition, the buffers discussed in this paper do not have return lanes, and future research can consider the problem of dynamic sequencing of buffers with return lanes.

Author Contributions: H.Z. wrote the manuscript and collected the data. W.D. collected the data and contributed to the writing of the text. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available on request from the corresponding author. Restrictions apply to the availability of these data, which were used under license for this study.

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

References

1. Boysen, N.; Fliedner, M.; Scholl, A. Sequencing mixed-model assembly lines: Survey, classification and model critique. *Eur. J. Oper. Res.* **2009**, *192*, 349–373. [CrossRef]
2. Baybars, I. A survey of exact algorithms for the simple assembly line balancing problem. *Manag. Sci.* **1986**, *32*, 909–932. [CrossRef]
3. Becker, C.; Scholl, A. A survey on problems and methods in generalized assembly line balancing. *Eur. J. Oper. Res.* **2006**, *168*, 694–715. [CrossRef]
4. Yurtkuran, A.; Yagmahan, B.; Emel, E. A novel artificial bee colony algorithm for the workforce scheduling and balancing problem in sub-assembly lines with limited buffers. *Appl. Soft Comput.* **2018**, *73*, 767–782. [CrossRef]
5. Zhu, X.; Hu, S.J.; Koren, Y.; Marin, S.P. Modeling of Manufacturing Complexity in Mixed-Model Assembly Lines. *J. Manuf. Sci. Eng.* **2008**, *130*, 649–659. [CrossRef]
6. Takai, S. An Approach to Integrate Product and Process Design Using Augmented Liaison Diagram, Assembly Sequencing, and Assembly Line Balancing. *J. Mech. Des.* **2021**, *143*, 1–24. [CrossRef]
7. Lahmar, M.; Ergan, H.; Benjaafar, S. Resequencing and feature assignment on an automated assembly line. *IEEE Trans. Robot. Autom.* **2003**, *19*, 89–102. [CrossRef]
8. Cordeau, J.F.; Laporte, G.; Pasin, F. Iterated tabu search for the car sequencing problem. *Eur. J. Oper. Res.* **2008**, *191*, 945–956. [CrossRef]
9. Gagné, C.; Gravel, M.; Price, W.L. Solving real car sequencing problems with ant colony optimization. *Eur. J. Oper. Res.* **2006**, *174*, 1427–1448. [CrossRef]
10. Mansouri, S.A. A multi-objective genetic algorithm for mixed-model sequencing on JIT assembly lines. *Eur. J. Oper. Res.* **2005**, *167*, 696–716. [CrossRef]
11. Prandtstetter, M.; Raidl, G.R. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *Eur. J. Oper. Res.* **2008**, *191*, 1004–1022. [CrossRef]
12. McMullen, P.R.; Tarasewich, P.; Frazier, G.V. Using genetic algorithms to solve the multi-product JIT sequencing problem with set-ups. *Int. J. Prod. Res.* **2000**, *38*, 2653–2670. [CrossRef]
13. Boysen, N.; Scholl, A.; Wopperer, N. Resequencing of mixed-model assembly lines: Survey and research agenda. *Eur. J. Oper. Res.* **2012**, *216*, 594–604. [CrossRef]
14. Wortmann, D.; Spieckermann, S. Manufacturing line simulation of automotive industry to enhance productivity and profitability. *Automot. Simul.* **1995**, *95*, 91–106.
15. Parrello, B.D.; Kabat, W.C.; Wos, L. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *J. Autom. Reason.* **1986**, *2*, 1–42. [CrossRef]
16. Kis, T. On the complexity of the car sequencing problem. *Oper. Res. Lett.* **2004**, *32*, 331–335. [CrossRef]
17. Solnon, C.; Nguyen, A.; Artigues, C.; Van Cung, D. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *Eur. J. Oper. Res.* **2008**, *191*, 912–927. [CrossRef]
18. Drexl, A.; Kimms, A. Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. *Manag. Sci.* **2001**, *47*, 480–491. [CrossRef]
19. Thiruvady, D.; Morgan, K.; Amir, A.; Ernst, A.T. Large neighbourhood search based on mixed integer programming and ant colony optimisation for car sequencing. *Int. J. Prod. Res.* **2020**, *58*, 2696–2711. [CrossRef]
20. Estellon, B.; Gardi, F.; Nouioua, K. Two local search approaches for solving real-life car sequencing problems. *Eur. J. Oper. Res.* **2008**, *191*, 928–944. [CrossRef]
21. Gavranović, H. Local search and suffix tree for car-sequencing problem with colors. *Eur. J. Oper. Res.* **2008**, *191*, 972–980. [CrossRef]

22. Siala, M.; Hebrard, E.; Huguet, M.J. A study of constraint programming heuristics for the car-sequencing problem. *Eng. Appl. Artif. Intell.* **2015**, *38*, 34–44. [CrossRef]
23. Bulgak, A. Analysis and design of split and merge unpaced assembly systems by metamodelling and stochastic search. *Int. J. Prod. Res.* **2006**, *44*, 4067–4080. [CrossRef]
24. Muhl, E.; Charpentier, P.; Chaxel, F. Optimization of physical flows in an automotive manufacturing plant: Some experiments and issues. *Eng. Appl. Artif. Intell.* **2003**, *16*, 293–305. [CrossRef]
25. Spieckermann, S.; Gutenschwager, K.; Voß, S. A sequential ordering problem in automotive paint shops. *Int. J. Prod. Res.* **2004**, *42*, 1865–1878. [CrossRef]
26. Yu, D.Y.; Zhang, S.Q.; Shao, X.Y.; Liu, S.Q.; Tian, Y.H. A gravity-like mechanism for car sequencing problem with multistage sequencing buffer. In *Advanced Materials Research*; Trans Tech Publications: Stafa-Zurich, Switzerland, 2011; Volume 314, pp. 2232–2237. [CrossRef]
27. Moon, D.H.; Song, C.; Ha, J.H. A dynamic algorithm for the control of automotive painted body storage. *Simulation* **2005**, *81*, 773–787. [CrossRef]
28. Pereira, J.; Vilà, M. An exact algorithm for the mixed-model level scheduling problem. *Int. J. Prod. Res.* **2015**, *53*, 5809–5825. [CrossRef]
29. Son, B.; Kim, J.W.; Lee, D.; Jung, S.Y. Genetic algorithm with species differentiation based on kernel support vector machine for optimal design of wind generator. *IEEE Trans. Magn.* **2019**, *55*, 1–4. [CrossRef]
30. Schoen, M.P.; Hoover, R.C.; Chinvorarat, S.; Schoen, G.M. System identification and robust controller design using genetic algorithms for flexible space structures. *J. Dyn. Syst. Meas. Control* **2009**, *131*, 304–314. [CrossRef]
31. Xidias, E.; Moulianitis, V.; Azariadis, P. Optimal robot task scheduling based on adaptive neuro-fuzzy system and genetic algorithms. *Int. J. Adv. Manuf. Technol.* **2021**, *115*, 927–939. [CrossRef]
32. Knust, J.; Podszus, F.; Stonis, M.; Behrens, B.A.; Overmeyer, L.; Ullmann, G. Preform optimization for hot forging processes using genetic algorithms. *Int. J. Adv. Manuf. Technol.* **2017**, *89*, 1623–1634. [CrossRef]
33. Zhengdong, Z.; Yuanhua, C.; Dongdong, W.; Zili, Y. Reconstruction of the linac photon spectrum based on prior knowledge and the genetic algorithm. *J. Southeast Univ. (Engl. Ed.)* **2014**, *30*, 311–314.
34. Singh, P.; Singh, R.K.; Joshi, D.; Bathla, G. Knowledge Application to Crossover Operators in Genetic Algorithm for Solving the Traveling Salesman Problem. *Int. J. Softw. Innov. (IJSI)* **2022**, *10*, 1–20. [CrossRef]
35. Mehta, C.; Patil, L.; Dutta, D. An approach to determine important attributes for engineering change evaluation. *J. Mech. Des.* **2013**, *135*, 041003. [CrossRef]
36. Malhan, R.; Jomy Joseph, R.; Bhatt, P.M.; Shah, B.; Gupta, S.K. Algorithms for improving speed and accuracy of automated three-dimensional reconstruction with a depth camera mounted on an industrial robot. *J. Comput. Inf. Sci. Eng.* **2022**, *22*, 031012. [CrossRef]
37. Gottlieb, J.; Puchta, M.; Solnon, C. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In Proceedings of the Applications of Evolutionary Computing: EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, London, UK, 14–16 April 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 246–257.