Applied Mathematical Modelling 64 (2018) 285–297

Contents lists available at ScienceDirect

Applied Mathematical Modelling

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

Accelerated dynamic programming algorithms for a car resequencing problem in automotive paint shops

Sungwon Hong[a], Jinil Han[b], Jin Young Choi[c], Kyungsik Lee[a,*]

[a] Department of Industrial Engineering & Institute for Industrial Systems Innovation, Seoul National University, Seoul, Republic of Korea
[b] Department of Industrial and Information Systems Engineering, Soongsil University, Seoul, Republic of Korea
[c] Department of Industrial Engineering, Ajou University, Suwon, Republic of Korea

ARTICLE INFO

Article history:
Received 3 November 2017
Revised 24 May 2018
Accepted 24 July 2018
Available online 3 August 2018

Keywords:
Car resequencing problem
Automotive paint shops
Dynamic programming
Heuristic algorithm

ABSTRACT

In this paper, a car resequencing problem (CRP) for automotive paint shops is considered, whereby a set of cars conveyed from an upstream shop to one of the multiple conveyors is retrieved sequentially before the painting operation. The aim of the CRP is to find a car retrieval sequence that minimizes the sequence-dependent changeover cost, which is the cost that is incurred when two consecutive cars do not share the same color. For this problem, we propose accelerated dynamic programming (ADP) algorithms that utilize strong combinatorial lower bounds and effective upper bounds in a standard dynamic programming framework, thus outperforming existing exact algorithms. Testing of our algorithms over a wide range of instances confirmed that they are more efficient than the existing approaches and are also more applicable in practice.

1. Introduction

A typical automotive manufacturing process comprises four major shops: a press shop, a body shop, a paint shop, and an assembly shop. The entire production process needs to be managed in a way that improves operational efficiency. One of the common means of controlling the production process is to use a storage system. In particular, the multiple-conveyor buffer system shown in Fig. 1 is the most widely employed in the automotive industry due to its simplicity and relatively low cost. It is commonly positioned between the body shop and the paint shop in order to link the two shops. A group of cars leaving a body shop forms a line in a conveyor waiting to be stored in one of the upstream conveyors. At the same time, one of the cars in the upstream conveyors is retrieved to be loaded to a downstream conveyor heading to a paint shop. The fundamental utility of this system is its control of retrieval sequences, specifically the resequencing of cars in such a way as to reduce the sequence-dependent changeover costs in the paint shop. This type of storage system, however, offers only a limited retrieval-sequence-formation flexibility, since retrievals from each individual conveyor must be performed on a first-in-first-out (FIFO) basis, which incurs precedence relations among the cars.

A changeover cost in the paint shop is incurred when the colors of two consecutive incoming cars are not identical. In this case, the spray nozzles must be flushed out using solvent, preparatory to painting the next car, in order to prevent color mixing. The changeover cost can vary depending on the chemical composition of the paint being changed. This means that the changeover cost can vary depending on the sequence of retrieval from the storage system [1]. The changeover

* Corresponding author.
E-mail address: optima@snu.ac.kr (K. Lee).

https://doi.org/10.1016/j.apm.2018.07.035
0307-904X/

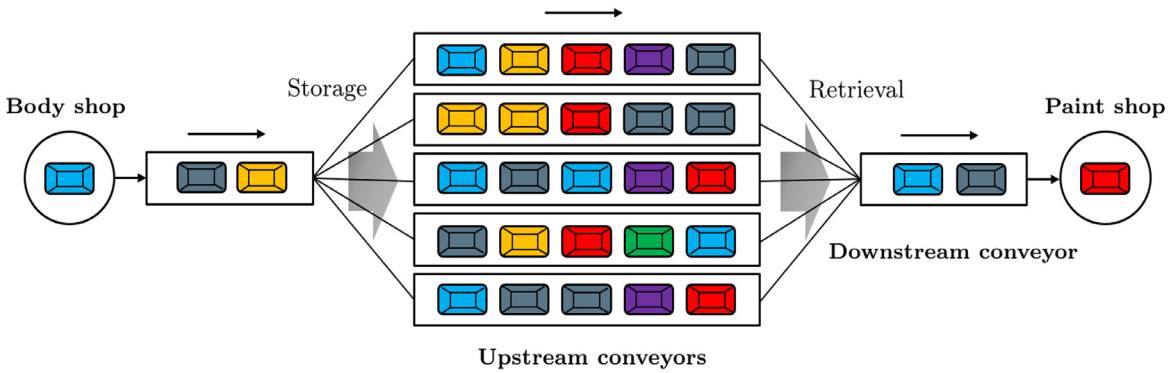


Fig. 1. Storage system in automotive paint shops.

cost incurred in paint shops is significantly high in practice. For example, Moon et al. [2] showed a case in which the annual changeover costs in the paint shop amount to $5.3 million under the assumption of producing 240,000 cars per year. Spieckermann et al. [3] also describes a case where an annual cost of $2.7 million is incurred to produce 200,000 cars per year. Therefore, resequencing cars in order to minimize the sequence-dependent changeover costs in a storage system is important to reduce the overall manufacturing costs for car manufacturers.

In this paper, we consider a car resequencing problem (CRP) of an automotive paint shop. The car retrieval sequences are determined from the multiple-conveyor storage system shown in Fig. 1 in order to minimize the changeover cost in the paint shop while satisfying the precedence relations among cars.

The CRP is known to be $\mathcal{NP}$-hard [4,5], even for the special case in which the changeover costs are assumed to be the same regardless of car colors. This case is referred to as the Number of Changeovers (NC) case, where the objective of the CRP is simply to minimize the number of changeovers. For the general case, which we refer to as the General Changeover cost (GC) case, the CRP has proven to be $\mathcal{NP}$-hard even if each upstream conveyor has only one car [6].

While a number of related studies have been carried out, most of these focused on the special case where the objective is to minimize the number of color changes [3–5,7–9]. Only a few studies have been carried out on exact solution algorithms. Spieckermann et al. [3] proposed a branch-and-bound (B&B) algorithm in an attempt to reduce the search space by means of a simple lower bound. They also compared the proposed algorithm with a simple rule that is a widely used heuristic in practice. Although the solution quality of the proposed algorithm is superior to that of the simple rule, its computational time is highly unstable as the number of cars increases. Epping et al. [4,5] proposed another exact algorithm based on the A* algorithm for the multiple sequence alignment problem [10]. However, in their computational results, the algorithm proved impractical for many conveyors, since the complexity increases exponentially with the number of conveyors.

In several other studies, heuristic algorithms have been proposed. While Ding and Sun [7] proposed simple heuristic procedures, their performance was not compared with that of other existing exact and heuristic algorithms. Hartmann and Runkler [8] devised a heuristic procedure based on ant colony optimization (ACO) and compared it with the existing approaches. Although the number of changeovers was reduced, the computational time was impractical. Sun et al. [9] proposed two heuristic procedures to deal with storing and retrieval sequences, respectively. Their computational results indicated performance comparable to that of the existing B&B algorithm proposed by Spiekermann et al. [3].

To the best of our knowledge, few studies have been carried out on the CRP with a sequence-dependent changeover cost, even though some studies consider sequence-dependent cost structure in different contexts [11–13]. Spieckermann et al. [3] was the first to include sequence-dependent changeover cost in their CRP formulation, but they proposed a branch-and-bound algorithm only for the NC type problem. Recently, Ko et al. [6] considered the CRP with sequence-dependent changeover cost. They proposed a dynamic programming algorithm and a genetic algorithm (GA)-based heuristic. However, they did not test the performance of their dynamic programming algorithm computationally. Rather, they tested their GA-based heuristic by comparing it with optimal solutions obtained by solving integer programs for small-sized instances. However, their GA-based heuristic was not fully investigated for large-sized instances, due to the lack of any scalable exact algorithm for the CRP.

As noted above, the CRP literature is very limited, especially for the GC type. Even though a few studies have been carried out on the NC type, the number of studies on efficient exact algorithms is also limited. Moreover, some authors initially proposed dynamic programming algorithms but abandoned this approach before completing any thorough investigation of computational performance. In an attempt to fill these gaps in the literature, we propose accelerated dynamic programming (ADP) algorithms that incorporate strong combinatorial lower bounds and effective upper bounds into a standard dynamic programming framework, thereby leading to a significant reduction of the search space and computational time. The computational results indicate a comparable computation time for the proposed ADP algorithms relative to that of the GA-based heuristic algorithm [6].

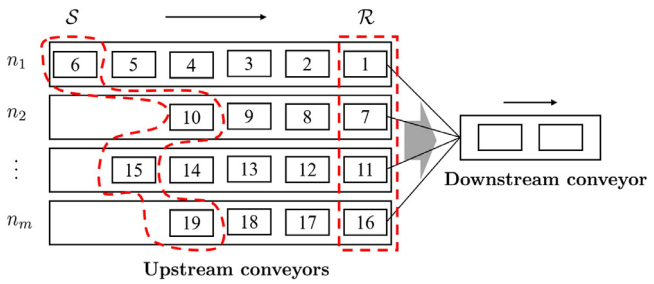


Fig. 2. Configuration of storage system.

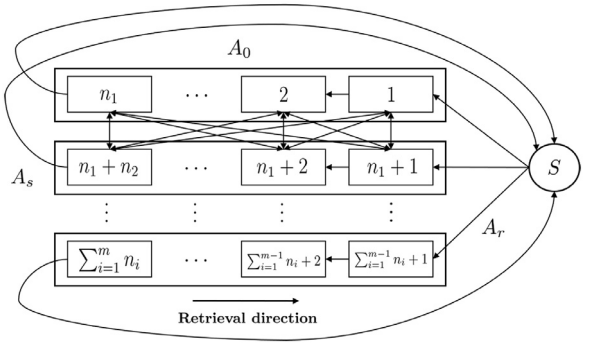


Fig. 3. Network representation of CRP.

This paper is organized as follows. Section 2 describes the CRP and introduces its integer program. Section 3 introduces a dynamic programming algorithm for solving the CRP and proposes acceleration techniques including strong lower and upper bounds to increase its speed. Section 4 presents the computational results, and Section 5 discusses the conclusions.

2. Problem definition and integer programming formulation

In this section, we first present the formal definition of the CRP, and then introduce an integer programming formulation for the proposed CRP. Fig. 2 illustrates a storage system consisting of a set of upstream conveyors $M = \{1, \ldots, m\}$ and a set of cars $N = \{1, \ldots, n\}$. Upstream conveyor $k \in M$ has n_k cars, such that $n = \sum_{k \in M} n_k$ and $\sum_{i=1}^{k-1} n_i + j$ is the index of the jth car on conveyor k. Since each upstream conveyor operates on a FIFO basis, the pth car on the upstream conveyor k should be released to the downstream conveyor before the qth car on the same upstream conveyor whenever $p < q$. Note that there is no precedence relation among cars on different upstream conveyors. Let $\mathcal{R}$ be the set of cars at the right-most position on each conveyor, i.e., $\mathcal{R} = \{j \in N \mid j = \sum_{i=1}^{k-1} n_i + 1,\ \forall k \in M\}$, and let $\mathcal{S}$ be a set of cars on the left-most position, i.e., $\mathcal{S} = \{j \in N \mid j = n_k + \sum_{i=1}^{k-1} n_i,\ \forall k \in M\}$. Note that each time a car is retrieved from the upstream conveyors to the downstream conveyor, one of the cars in $\mathcal{R}$ is retrieved. Let R_k and S_k refer to the cars at the right- and left-most positions on conveyor k, respectively. When two consecutive cars i and j released to the downstream conveyor do not share the same color, a changeover cost c_{ij} is incurred. On the other hand, when the colors of cars i and j are identical, no changeover cost is incurred. We assume that $c_{ij} + c_{jk} \geq c_{ik}$ for any three different cars i, j, and k. The problem then becomes determining the retrieval sequence that minimizes the total changeover cost.

Prior to developing an integer program for the CRP, let us first define a directed network. Let V_0 be the set of nodes, each of which corresponds to each car on upstream conveyors. Then, we define the set of directed arcs A_0, each (i, j) of which exists if car i can be retrieved immediately before car j. The cost of each arc (i, j), c_{ij}, is equal to the changeover cost from car i to j. Here, we introduce a dummy node s along with two sets of dummy arcs $A_r = \{(s, j) \mid j \in \mathcal{R}\}$ and $A_s = \{(j, s) \mid j \in \mathcal{S}\}$, where all arc costs are 0. Then, the CRP is equivalent to finding a minimum cost tour on a directed network $G = (V, A)$, where $V = \{s\} \cup V_0$ and $A = A_0 \cup A_r \cup A_s$, as shown in Fig. 3. Using these notations, the CRP can be formulated as the following integer program:

$$\min \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{(i,j) \in A} x_{ij} = 1, \quad i \in V, \tag{2}$$

$$\sum_{(i,j)\in A} x_{ij} = 1, \quad j \in V, \tag{3}$$

$$u_i - u_j + |V|x_{ij} \le |V| - 1, \ \ i, j \in V, \ \ j \ne s, \tag{4}$$

$$u_s = 1, \tag{5}$$

$$2 \le u_i \le |V|, \quad i \in V, \quad i \ne s, \tag{6}$$

$$u_i \le u_{i+1}, \quad R_k \le i \le S_k - 1, \quad k \in M, \tag{7}$$

$$u_i \ge 0, \quad i \in V, \tag{8}$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in V. \tag{9}$$

The binary decision variable $x_{ij} = 1$ if an arc (i, j) is selected (i.e., car j is retrieved immediately after car i); otherwise 0 for all $(i, j) \in A$. The decision variable u_i denotes the position of car i in the tour. Constraints (2) and (3) imply that exactly one arc must be directed in and out of a car, respectively. Constraints (4) ensure that there should be no subtours. Constraint (5) states that the position of dummy node s must be 1. Constraints (6) ensure that the position of each car ranges from 2 to $|V|$. Constraints (7) enforce the precedence relation among the cars on the same upstream conveyor.

The above formulation is based on an integer program for the well-known traveling salesman problem (TSP) [14]. The linear programming (LP) relaxation of the above formulation gives poor lower bounds on the optimal objective value; thus, that formulation might not be a good choice for obtaining optimal solutions to the CRP, especially for large-sized instances. However, the bound of the LP relaxation can be strengthened by adding additional valid inequalities such as the following cut-set inequalities [14]. Given a nonempty proper subset S of V (i.e., $S \subset V$, $S \ne \emptyset$), the cut-set inequality is defined as

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \ge 1, \quad S \subset V, \ S \ne \emptyset, \tag{10}$$

where $\delta^+(S) = \{(i, j) \in A \mid i \in S, \ j \notin S\}$. Later, in the computational results, we show that adding these valid inequalities can significantly improve the bound of the LP relaxation of the above IP formulation. However, we also show that the lower bounds proposed in the following section are stronger and more efficiently computable than the strengthened LP relaxation.

3. Accelerated dynamic programming (ADP) algorithm

In this section, as a means of solving the CRP to optimality, we present an enhanced dynamic programming (DP) approach based on the DP algorithms proposed in the literature. Among those DP algorithms [6,15], we focus on that which was most recently published [6]. In general, DP algorithms are known to suffer from the curse of dimensionality as the problem size increases; indeed, it was for this reason that Ko et al. [6] did not perform the computational test for their DP algorithm. In order to alleviate the curse of dimensionality, we propose an ADP algorithm that utilizes effective bounding techniques to help reduce the search space. We first describe the DP algorithm proposed by [6], which we call the basic DP algorithm, with the help of a state transition network.

3.1. Basic dynamic programming algorithm

Let us first define a state transition network by which the DP algorithm can be easily described. State S is defined by an $(m+1)$-tuple $(P_1, P_2, \ldots, P_m, k)$, where P_i is the number of retrieved cars from conveyor i, and k is the index of the conveyor from which the last car was retrieved. Note that the index of the last retrieved car at a state $(P_1, P_2, \ldots, P_m, k)$ is $\sum_{i=1}^{k-1} n_i + P_k$. A transition refers to the event that a car is retrieved from one of the m conveyors. More precisely, if the current state is $S = (P_1, \ldots, P_l, \ldots, P_m, k)$, and a car is retrieved from conveyor l, this event corresponds to the transition from state $S = (P_1, \ldots, P_l, \ldots, P_m, k)$ to $S' = (P_1, \ldots, P_l + 1, \ldots, P_m, l)$. We also associate cost $c(S, S')$ with this transition, which is the changeover cost between two cars $\sum_{i=1}^{k-1} n_i + P_k$ and $\sum_{i=1}^{l-1} n_i + P_l + 1$. The states can then be grouped according to the number of retrieved cars, $\Sigma_{i\in M} P_i$, which we call a stage. That is, state $S = (P_1, P_2, \ldots, P_m, k)$ is at stage $\Sigma_{i\in M} P_i$. Note that transitions always occur between a state at stage v and another state at stage $v + 1$. Finally, we add the initial state $s = (0, \ldots, 0, 0)$ that is connected to the states at stage 1 via corresponding transitions, and we add the final state $t = (n_1, n_2, \ldots, n_m, 0)$ to

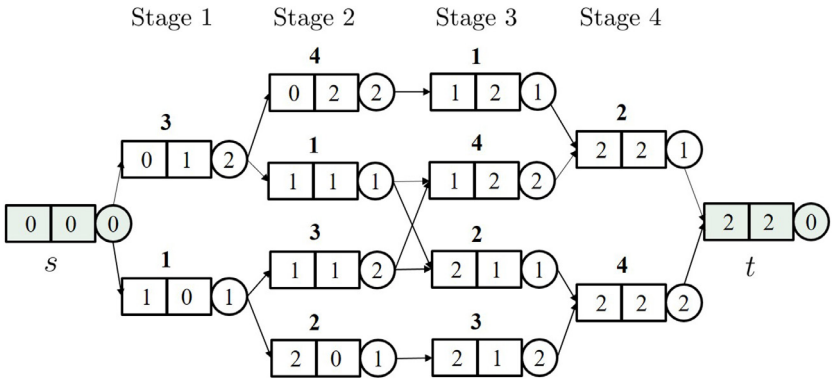


Fig. 4. State transition network.

which the states at stage $\Sigma_{i\in M} n_i$ are connected. Fig. 4 graphically illustrates an example of the state transition network for the case where each of the two conveyors contains two cars.

In the state transition network defined above, a path from state s to state t constitutes a feasible car retrieval sequence; therefore, the optimal solution to the CRP corresponds to the shortest path from state s to state t. Note that the state transition network is a directed acyclic graph, which enables us to find the shortest path in linear time, $O(|N| + |A|)$. It is sufficient to go through all of the states stage by stage until we arrive at state t. The basic DP algorithm is described in Algorithm 1.

Algorithm 1: Basic Dynamic Programming Algorithm for Solving CRP.

$cost(s) = 0$;
foreach *stage* v *in* $\{1, \ldots, \sum_{i=1}^{m} n_i + 1\}$ **do**
  foreach *state* S *at stage* v **do**
    $cost(S) = \min_{S':\exists S' \to S}\{cost(S') + c(S', S)\}$
  end
end

However, the number of states in a state transition network grows exponentially as the number of conveyors m increases. This is because the number of possible states is $O(m\prod_{k=1}^{m} n_k)$, and the running time of the basic DP is therefore bounded by $O(m^2\prod_{k=1}^{m} n_k)$, the maximum number of transitions in the network. In order to avoid a rapid increase of the number of states in the course of the basic DP algorithm and, thereby, accelerate the overall algorithm, unnecessary states need to be eliminated by computing tight lower bounds for current states and comparing them with an effective upper bound. In the following subsections, we present the overall procedure of the ADP algorithm, and then present several lower bounds and upper-bounding procedures that can be integrated into the ADP algorithm.

3.2. Accelerated dynamic programming (ADP) algorithm

The basic DP algorithm can be accelerated with the help of lower and upper bounds; that is, we can eliminate unnecessary states in the course of the DP algorithm by computing lower bounds for each state and comparing them with an upper bound. Let UB be the global upper bound on the optimal value of the CRP, and let $LB(S)$ be the lower bound on the changeover cost for the remaining cars at state S. Then, we can prune the current state S if $E(S) := cost(S) + LB(S)$ is greater than or equal to UB. The pseudo-code, given in Algorithm 2, describes the overall procedure of the proposed ADP algorithm.

The lower bounds $\mathbf{LB}_1$, $\mathbf{LB}_2$, and $\mathbf{LB}_3$ given in the following Section 3.2.1 are used to compute the $LB(S)$ for each state S. In general, the upper bounds of the optimization problem are computed by finding feasible solutions, since the costs of feasible solutions are greater than or equal to the optimal value. In Algorithm 2, the initial upper bound $\mathbf{UB}_g$ is computed by the heuristic algorithm given in Section 3.2.2. The upper bound $UB(S)$ for each state S is computed using the following simple heuristic to reduce the computational burden:

- Select the car with the same color as the one most recently retrieved from the conveyors.
- If there is no such car at the retrieval point, choose the car that incurs the least changeover cost.

In each state S, this simple heuristic is applied to find a feasible retrieval sequence for the remaining cars at state S. Then, $UB(S)$ is set to the cost of the obtained retrieval sequence. If $cost(S) + UB(S) < UB$, the global upper bound UB is updated to $cost(S) + UB(S)$.

Algorithm 2: Accelerated Dynamic Programming Algorithm for Solving CRP.

```
UB := UB_g;
cost(s) := 0;
foreach stage v in {1, ..., Σ_{i=1}^{m} n_i + 1} do
    foreach state S at stage v do
        cost(S) := min_{S': ∃S' → S}{cost(S') + c(S', S)};
        compute LB(S) and UB(S);
        E(S) := cost(S) + LB(S);
        if cost(S) + UB(S) < UB then
            UB := cost(S) + UB(S);
        end
        if E(S) ≥ UB then
            prune state S;
        end
    end
end
```

3.2.1. Computing lower bounds

One of the well-known methods for computing the lower bounds of the given optimization problem is to solve the relaxations of the original problem. In this subsection, we introduce lower bounds ($\mathbf{LB}_1$, $\mathbf{LB}_2$, and $\mathbf{LB}_3$) that are more efficiently computable than the LP relaxations of the IP formulation given in Section 2.

Without loss of generality, we can assume that two consecutive cars on the same conveyor have different colors, because two consecutive cars having the same color can be retrieved consecutively without increasing the total changeover cost and therefore can be regarded as a single car.

The set of constraints that make the CRP difficult to solve is (7), which is the precedence relations among cars on the same conveyor. Note that the CRP becomes much easier if we remove such precedence relations. Therefore, we derive lower bounds for the CRP by investigating the problem in the absence of the precedence relations. Let C be the set of all colors with which cars on the conveyors should be painted, and let γ be the minimum changeover cost between two different colors. The following proposition shows that the first lower bound ($\mathbf{LB}_1$) is valid.

Proposition 1. *Let z^* be the optimal value of the* CRP, *we then have*

$$z^* \geq \gamma(|C| - 1).$$

Proof. Let w^* be the optimal value of the CRP in the absence of precedence relations. Then, $z^* \geq w^*$, since w^* is the optimal value of a relaxation of the CRP. It is therefore sufficient to show that $w^* \geq \gamma(|C| - 1)$. Now, suppose that the precedence relations are all relaxed. Note that we assumed that the triangle inequality holds between three different colors; i.e., the changeover cost between color p and color r is not greater than the sum of the changeover cost between color p and color q and the changeover cost between color q and color r. Under this assumption, the optimal retrieval sequence should retrieve all cars that have the same color at the same time and then switch to another color. Then, since the number of color changes equals $|C| - 1$, the inequality $w^* \geq \gamma(|C| - 1)$ directly follows. □

The lower bound $\mathbf{LB}_1$ can be strengthened by unrelaxing the precedence relations among the cars on one of the m conveyors. For conveyor k, let $C(k)$ be the set of colors that is assigned to the cars and n_{ki} be the number of cars that have color i. Also, let v_i be the minimum changeover cost among all changeover costs from all colors except for color i to color i. Then, a strengthened lower bound $\mathbf{LB}_2$ can be derived as defined in the following proposition:

Proposition 2. *Let z^* be the optimal value of the* CRP, *we then have*

$$z^* \geq \max_{k \in M}\{\gamma(|C| - 1) + \sum_{\{i \in C(k) | n_{ki} > 1\}} v_i(n_{ki} - 1)\}.$$

Proof. Let w_k^* be the optimal value of the CRP with the precedence relations for all of the conveyors, but with k relaxed. Then, $z^* \geq w_k^*$ for all $k \in M$, which implies $z^* \geq \max_{k \in M}\{w_k^*\}$. Therefore, it is sufficient to show that

$$w_k^* \geq \gamma(|C| - 1) + \sum_{\{i \in C(k) | n_{ki} > 1\}} v_i(n_{ki} - 1)$$

for conveyor k. Now, suppose that the precedence relations among cars on all conveyors except for k are relaxed. If conveyor k contains at least two nonconsecutive cars with the same color, additional color changes are necessary in addition to the color changes needed when all of the precedence relations are relaxed. Note that the number of additional color changes corresponding to color i is equal to $n_{ki} - 1$. Then, the additional changeover cost incurred by unrelaxing the precedence relations for conveyor k is at least $\sum_{\{i \in C(k) | n_{ki} > 1\}} v_i(n_{ki} - 1)$, which directly leads to the above inequality. □

The lower bound $\mathbf{LB}_2$ can be further improved for the CRP of the GC type. For conveyor k for which the precedence relations are not relaxed, if at least two nonconsecutive cars have the same color i, all such cars except for the first car contribute to the total changeover cost. Let $B(k, i)$ be such a set. Note that if any cars not on conveyor k have color $i \in C(k)$, they can be merged into the car on conveyor k of the same color i, since this does not increase the total changeover cost. Therefore, any cars that can be retrieved immediately before $j \in B(k, i)$ should have colors that are either not in the set $C(k)$ or are the same as that of car $j-1$. Let $H(j)$ be the set of cars that can be retrieved immediately before j. The following proposition thus directly follows, and we let $\mathbf{LB}_3$ be the lower bound obtained from this proposition. Note that for the CRP of the NC type, $\mathbf{LB}_2$ and $\mathbf{LB}_3$ are the same.

Proposition 3. *Let z^* be the optimal value of the* CRP, *we then have*

$$z^* \geq \max_{k \in M}\{\gamma(|C|-1) + \sum_{\{i \in C(k) | n_{ki} > 1\}} \sum_{j \in B(k,i)} \min_{j' \in H(j)}\{c_{j'j}\}\}.$$

In Algorithm 2, the lower bound $LB(S)$ for each state S is the sum of the minimum changeover cost between the last processed car and the cars in $\mathcal{R}$ and one of the three lower bounds ($\mathbf{LB}_1$, $\mathbf{LB}_2$, and $\mathbf{LB}_3$).

3.2.2. Computing initial upper bounds

The proposed ADP algorithm needs a strong initial upper bound ($\mathbf{UB}_g$) to help significantly reduce the search space. To achieve this, we devised an efficient heuristic algorithm, which we call the *Approximate Bounding* (AB) heuristic, to obtain a good feasible solution to the CRP. The basic idea of the AB heuristic is to manage a small number of promising states at each stage. The detailed procedure of the AB heuristic is given in Algorithm 3 .

Algorithm 3: Approximate Bounding (AB) Heuristic Algorithm.

```
UB := UB_0;
cost(s) := 0;
foreach stage v in {1, ..., Σ_{i=1}^{m} n_i + 1} do
    m_v := ∞;
    foreach state S at stage v do
        cost(S) := min_{S': ∃S' → S}{cost(S') + c(S', S)};
        compute LB(S) and UB(S);
        E(S) := cost(S) + LB(S);
        if E(S) < m_v then
            m_v := E(S);
        end
    end
    foreach state S at stage v do
        if cost(S) + UB(S) < UB then
            UB := cost(S) + UB(S);
        end
        if E(S) > m_v + (γ × σ) then
            prune state S;
        end
    end
end
```

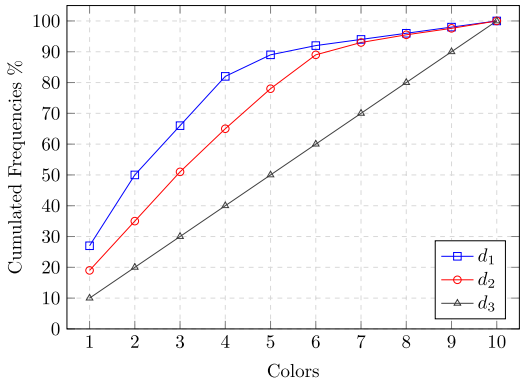
At each stage v, we compute the minimum value of $E(S)$ among the states; let this value be m_v. Then, we prune state S whenever $E(S)$ is greater than $m_v + (\gamma \times \sigma)$, where σ is a control parameter. $\mathbf{UB}_0$ and $UB(S)$ for each state S are computed by using the same simple heuristic used in the ADP algorithm. Note that the optimality of the obtained solution is not guaranteed, since we can prune the state that can lead to optimal solutions. If we increase the value of σ, the number of states to visit increases, thus improving the solution quality, while the computation time increases.

Of course, the AB heuristic can be used as a standalone heuristic algorithm for the CRP. In the following section, we show that the AB heuristic algorithm finds good solutions within a very short time.

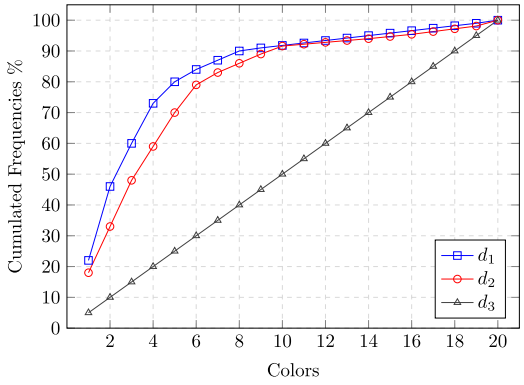
4. Computational results

4.1. Test instances

We conducted several computational tests on randomly generated instances to demonstrate the performance of the proposed algorithms. First, we designed different storage-system configurations by changing the number of upstream conveyors



(a) 10 colors



(b) 20 colors

Fig. 5. Color distributions used in computational tests.

and the number of cars on each conveyor. We assumed that, without loss of generality, each conveyor contains the same number of cars. Specifically, we considered four configurations (the number of conveyors multiplied by the number of cars), namely 5×6, 7×8, 10×3, and 3×10. Note that 5×6 and 7×8 simulate real-world practice, where the numbers of conveyors and cars are almost equal [2,3], and 10×3, 3×10 are used to consider unusual cases wherein the number of conveyors is much larger than that of cars or vice versa. The numbers of colors of cars are given as 10 and 20, and the colors are assumed to follow the distribution d_1, d_2 and d_3 depicted in Fig. 5. Note that d_1 and d_2 simulate real-world practice, where most cars are painted in only a few colors; meanwhile, d_3 is the uniform distribution used to reflect the case wherein all of the colors are chosen with the same preference [3,9]. For each combination of storage-system configuration, number of colors and color distribution, we randomly generated 20 instances. For each instance, two types of CRPs, namely the NC and GC types, were considered in the computational tests. For the NC type, the changeover cost was always given as 1 regardless of the two colors, while for the GC type, the changeover costs between different colors were randomly generated within the range of 10 to 20 ($10 \le c_{ij} \le 20$). This is because the average value of the changeover costs usually does not exceed twice the minimum value in real-world automotive manufacturing factories [1]. Finally, each instance was preprocessed by merging consecutive cars of the same color on the same conveyor into a single car. Note that this preprocessing does not increase optimal changeover costs, as mentioned in the previous section.

Our algorithms were coded in C#, and all of the tests were performed on an Intel Core 3.10 GHz PC with 16GB RAM. We used Xpress 8.0 [16] to solve the LP relaxations of formulations (1)–(9) given in Section 2.

4.2. Test results for NC instances

The performance of the proposed algorithm was initially evaluated for NC instances. We first compared the lower bounds obtained by the proposed relaxation techniques. Table 1 shows the results. Note that $\mathbf{LB}_3$ is omitted because it gives the same bound as $\mathbf{LB}_2$. In addition to $\mathbf{LB}_1$ and $\mathbf{LB}_2$, the optimal value of the LP relaxation of formulations (1)–(9) (**LPR**) and the optimal value improved by adding cut-set inequalities (10) to the LP relaxation ($\mathbf{LPR}_{\text{cut}}$) are also reported. In the table, the headings *opt, obj*, and *time* refer to the optimal objective values of the instance, the objective value, and the computational time (in seconds) computed for each lower bound, respectively. Note that each row refers to the average value

Table 1
Comparison of lower bounds on NC instances.

Instance				$\mathbf{LB}_1$	$\mathbf{LB}_2$	LPR		$\mathbf{LPR}_{cut}$	
config.	*#color*	*dist.*	*opt*	*obj*	*obj*	*obj*	*time*	*obj*	*time*
5×6	10	d_1	11.4	5.9	8.3	2.4	0.07	6.0	0.49
		d_2	12.7	6.8	8.6	2.6	0.09	7.1	0.54
		d_3	14.3	8.5	10.1	2.8	0.12	8.8	0.63
	20	d_1	12.2	6.4	8.8	3.2	0.07	6.7	0.53
		d_2	14.3	8.8	10.4	4.0	0.12	9.2	0.64
		d_3	18.7	14.8	15.8	8.8	0.21	15.5	0.74
	Average		13.9	8.5	10.3	4.0	0.11	8.9	0.59
7×8	10	d_1	17.4	7.1	10.7	2.1	0.71	7.3	2.67
		d_2	19.2	8.0	11.2	2.0	0.68	8.1	3.02
		d_3	22.1	9.0	11.8	1.4	0.78	9.2	3.54
	20	d_1	18.6	8.6	12.1	3.9	0.72	8.7	2.86
		d_2	22.4	11.1	14.0	3.9	0.87	11.3	3.24
		d_3	30.1	17.9	19.9	5.7	1.08	18.7	3.68
	Average		21.6	10.3	13.3	3.2	0.81	10.5	3.17
10×3	10	d_1	8.5	5.9	6.7	1.6	0.10	6.0	0.63
		d_2	9.8	6.8	7.6	1.6	0.10	6.8	0.64
		d_3	10.9	8.5	9.2	1.6	0.13	8.6	0.72
	20	d_1	9.1	6.4	7.2	2.0	0.10	6.4	0.56
		d_2	11.3	8.8	9.4	2.9	0.13	8.9	0.68
		d_3	16.2	14.8	15.2	7.3	0.23	15.1	0.86
	Average		11.0	8.5	9.2	2.8	0.13	8.6	0.68
3×10	10	d_1	14.6	5.9	10.1	4.0	0.04	6.7	0.37
		d_2	15.8	6.8	10.6	4.6	0.05	7.8	0.38
		d_3	17.8	8.5	11.8	5.6	0.10	9.6	0.44
	20	d_1	15.2	6.4	10.7	5.0	0.05	7.3	0.34
		d_2	17.2	8.8	12.3	5.9	0.09	9.8	0.44
		d_3	21.6	14.8	17.2	11.9	0.16	16.6	0.51
	Average		17.0	8.5	12.1	6.2	0.08	9.6	0.42

over 20 randomly generated instances. Note also that because $\mathbf{LB}_1$ and $\mathbf{LB}_2$ could be computed within a millisecond, their computational times are omitted from the table. The results demonstrate that $\mathbf{LB}_2$ yielded the best lower bound among those tested. Therefore, $\mathbf{LB}_2$ could be the best choice of the lower-bounding technique used in the ADP algorithm in terms of both the objective value and computation time. It can also be seen that **LPR** yields very poor lower bounds and needs more time to compute than $\mathbf{LB}_1$ or $\mathbf{LB}_2$. However, **LPR** can be improved significantly by adding cut-set inequalities (as shown in the table), although it is still worse than $\mathbf{LB}_2$.

Next, we evaluated the performance of the proposed upper-bounding heuristics by comparing with the existing heuristic algorithm in the literature, namely the GA of [6]. Note that the performance of the AB heuristic is dependent on the parameter σ. Therefore, we conducted preliminary tests to find an appropriate σ value that ensures the AB heuristic produces qualified feasible solutions within a short period of time. From the results, we chose $\sigma = 1$ and 2. The performance of our simple rule and AB heuristic in comparison with the existing GA is presented in Table 2. The *gap* in the table represents the absolute gap of the objective value obtained by a heuristic algorithm and a known optimal value (i.e., $gap = obj - opt$). We observed that the AB heuristic yields a very small gap, while the simple rule and the GA show relatively large gaps for all of the tested instances. Specifically, the AB heuristic with $\sigma = 2$ provides the best results in terms of objective values. The AB heuristic with both $\sigma = 1$ and 2 also runs faster than the GA. These results indicate that the AB heuristic with $\sigma = 2$ could be the best choice as a heuristic for solving the CRP.

We now show the computational performance of our ADP algorithm. Both $\mathbf{LB}_1$ and $\mathbf{LB}_2$ were used to compute the lower bounds at each state in the course of the ADP algorithm. Also, the initial upper bounds were computed using the AB heuristic with $\sigma = 2$. Table 3 shows a comparison of the performance of the ADP algorithm with that of the standard DP algorithm and branch-and-bound (B&B) algorithm [3]. In the table, ADP1 and ADP2 refer to the ADP algorithms that incorporate $\mathbf{LB}_1$ and $\mathbf{LB}_2$ for the lower-bound computation, respectively. In the B&B columns, *#opt* and *time(opt)* refer to the number of instances solved to optimality within the time limit and the average time needed to find the optimal solutions for solved instances, respectively. In the ADP1 and ADP2 columns, *%state* refers to the percentage of states generated during the execution of ADP1 and ADP2 relative to those generated during the execution of DP. The results are as follows. First, the DP and ADP algorithms were able to obtain optimal solutions for all of the tested instances within the time limit of one hour, while the B&B algorithm failed to solve some instances to optimality, in particular 7×8 instances known to be difficult. It can also be seen that the solution time of the B&B algorithm grows significantly as the number of colors increases or the color distribution approaches close to the uniform distribution. The DP algorithm shows very good performance for the small-sized instances (5×6 and 3×10), but the computational time increases rapidly as the problem size increases. The ADP algorithm, on the other hand, seems to be able to alleviate this curse of dimensionality, especially for large-sized instances (7×8 and 10×3). In particular, the solution-time difference for DP and ADP increases as the problem size increases. This reduction

Table 2
Comparison of upper-bounding heuristics on NC instances.

Instance			Simple rule	GA [6]		AB ($\sigma = 1$)		AB ($\sigma = 2$)	
config.	*#color*	*dist.*	*gap*	*gap*	*time*	*gap*	*time*	*gap*	*time*
5×6	10	d_1	5.6	0.1	3.87	0.1	0.02	0.0	0.08
		d_2	5.7	0.1	3.80	0.0	0.03	0.0	0.09
		d_3	6.5	0.1	3.73	0.1	0.03	0.0	0.08
	20	d_1	5.6	0.1	3.68	0.0	0.03	0.0	0.09
		d_2	6.1	0.2	3.69	0.0	0.03	0.0	0.10
		d_3	6.4	0.1	3.42	0.1	0.03	0.0	0.10
7×8	10	d_1	10.6	1.9	10.17	0.5	0.82	0.0	7.47
		d_2	10.7	2.3	10.35	0.7	0.91	0.2	9.77
		d_3	13.7	2.3	10.37	0.4	0.65	0.1	7.43
	20	d_1	11.5	1.8	10.01	0.3	1.10	0.0	9.77
		d_2	12.9	1.9	10.26	0.3	0.80	0.1	9.92
		d_3	14.3	2.2	9.58	0.7	0.49	0.1	4.86
10×3	10	d_1	3.2	0.3	4.34	0.0	0.47	0.0	1.09
		d_2	4.0	0.4	4.51	0.0	0.44	0.0	1.43
		d_3	5.1	0.6	4.53	0.0	0.30	0.0	0.88
	20	d_1	3.6	0.3	4.39	0.0	0.84	0.0	1.68
		d_2	4.5	0.4	4.48	0.0	0.66	0.0	1.68
		d_3	5.9	0.2	4.13	0.0	0.71	0.0	1.36
3×10	10	d_1	5.3	0.0	3.01	0.2	0.01	0.0	0.01
		d_2	5.2	0.0	3.01	0.1	0.01	0.0	0.01
		d_3	5.2	0.0	3.10	0.1	0.01	0.0	0.01
	20	d_1	5.0	0.0	2.94	0.0	0.01	0.0	0.01
		d_2	5.4	0.1	2.88	0.1	0.01	0.0	0.01
		d_3	4.6	0.1	2.72	0.1	0.01	0.0	0.01

Table 3
Performance comparison of exact algorithms on NC instances.

Instance			B&B [3]				DP[a]	ADP1[a]		ADP2[a]	
config.	*#color*	*dist.*	*gap*	*time*	*#opt*	*time(opt)*	*time*	*time*	*%state*	*time*	*%state*
5×6	10	d_1	0.0	2.27	20	1.28	0.15	0.11	32.1	0.08	8.9
		d_2	0.0	13.91	20	4.11	0.20	0.17	32.3	0.13	12.0
		d_3	0.0	12.60	20	11.39	0.22	0.12	19.9	0.10	7.9
	20	d_1	0.0	2.76	20	1.43	0.17	0.15	36.6	0.10	10.7
		d_2	0.0	11.14	20	6.71	0.19	0.16	26.3	0.13	10.2
		d_3	0.0	650.66	20	156.69	0.26	0.13	12.0	0.12	4.7
7×8	10	d_1	1.6	3,600[b]	7	891.54	78.75	50.21	53.2	16.16	12.9
		d_2	2.0	3,600[b]	3	1347.96	97.73	63.46	53.4	26.80	19.1
		d_3	4.1	3,600[b]	2	1021.34	116.92	64.16	47.5	29.86	20.7
	20	d_1	1.2	3,600[b]	2	756.06	86.09	69.39	53.0	23.06	14.5
		d_2	3.7	3,600[b]	3	463.31	114.65	73.58	43.1	30.80	16.9
		d_3	6.3	3,600[b]	1	227.46	153.91	42.14	23.3	22.90	11.3
10×3	10	d_1	0.0	0.58	20	0.21	31.06	2.74	7.6	1.27	3.5
		d_2	0.0	1.10	20	0.05	40.08	2.81	6.0	1.66	3.6
		d_3	0.0	4.87	20	2.42	46.71	1.86	3.5	1.26	2.3
	20	d_1	0.0	1.43	20	0.09	36.13	2.90	6.8	1.61	3.6
		d_2	0.0	17.16	20	9.56	42.79	2.87	5.6	2.00	3.8
		d_3	0.5	727.94	19	217.91	66.16	2.48	3.3	1.67	2.2
3×10	10	d_1	0.0	1.12	20	0.03	0.01	0.01	74.4	0.01	41.7
		d_2	0.0	1.13	20	0.06	0.01	0.02	72.2	0.01	46.2
		d_3	0.0	1.96	20	1.02	0.01	0.02	72.7	0.01	49.7
	20	d_1	0.0	1.09	20	0.07	0.01	0.02	77.1	0.01	42.7
		d_2	0.0	6.23	20	3.71	0.01	0.02	69.9	0.01	46.0
		d_3	0.0	30.45	20	10.05	0.01	0.02	59.1	0.01	39.4

[a] All instances were solved to optimality.
[b] Some instances failed to find optimal solution within one hour.

of solution time occurs because the ADP tends to explore a much smaller number of states during the execution of the algorithm, due to the tight lower- and upper-bounding techniques. The reduction of state space is significant, as seen in the *%state* columns. It was also observed that ADP2 shows better performance than ADP1, due to the use of a better lower bound.

Table 4
Comparison of lower bounds on GC instances.

Instance				$\mathbf{LB}_1$	$\mathbf{LB}_2$	$\mathbf{LB}_3$	LPR		$\mathbf{LPR}_{\text{cut}}$	
config.	*#color*	*dist.*	*opt*	*obj*	*obj*	*obj*	*obj*	*time*	*obj*	*time*
5×6	10	d_1	152.3	58.5	84.2	87.1	32.1	0.02	69.5	0.36
		d_2	165.9	68.0	87.8	89.2	33.4	0.02	80.8	0.42
		d_3	188.1	85.0	102.3	103.7	35.0	0.02	97.3	0.49
	20	d_1	162.2	64.3	89.2	92.2	44.2	0.02	77.1	0.39
		d_2	184.3	88.0	104.7	106.2	50.2	0.02	101.6	0.44
		d_3	231.9	148.0	157.6	158.0	101.7	0.03	165.3	0.49
	Average		180.8	85.3	104.3	106.0	49.4	0.02	98.6	0.43
7×8	10	d_1	231.5	71.0	110.8	113.7	30.3	0.33	82.8	2.54
		d_2	257.1	79.5	114.6	117.6	24.9	0.27	90.5	2.90
		d_3	285.5	90.0	120.3	122.5	16.3	1.07	101.8	4.20
	20	d_1	243.1	86.0	121.7	125.3	50.3	0.24	97.8	2.67
		d_2	286.6	111.0	140.7	142.7	48.8	0.19	122.0	3.43
		d_3	368.6	178.5	198.9	199.3	72.6	0.12	195.6	4.46
	Average		278.7	102.7	134.5	136.8	40.5	0.37	115.1	3.37
10×3	10	d_1	108.0	58.5	67.8	68.1	21.4	0.02	68.0	0.49
		d_2	120.9	68.0	76.9	77.0	20.1	0.02	75.4	0.58
		d_3	135.8	85.0	92.9	93.0	19.9	0.02	93.3	0.59
	20	d_1	115.2	64.3	72.1	73.2	27.7	0.02	72.7	0.58
		d_2	136.7	88.0	94.1	94.5	37.0	0.02	96.8	0.56
		d_3	188.4	148.0	151.5	151.6	82.6	0.03	157.9	0.58
	Average		134.2	85.3	92.5	92.9	34.8	0.02	94.0	0.56
3×10	10	d_1	199.7	58.5	104.4	113.6	55.2	0.01	82.1	0.27
		d_2	211.7	68.0	108.1	113.2	59.0	0.01	91.1	0.27
		d_3	241.8	85.0	120.7	124.4	71.0	0.02	108.8	0.30
	20	d_1	207.0	64.3	108.8	117.6	66.0	0.01	87.3	0.28
		d_2	234.3	88.0	123.7	127.7	74.2	0.02	111.9	0.31
		d_3	279.3	148.0	172.5	173.6	137.1	0.02	181.2	0.37
	Average		229.0	85.3	123.0	128.3	77.1	0.02	110.4	0.30

4.3. Test results for GC instances

In this subsection, we present the computational results for the GC instances. We first show the results of the lower-bound comparison in Table 4. The computational times of $\mathbf{LB}_1$, $\mathbf{LB}_2$, and $\mathbf{LB}_3$ are omitted from the table, since they were less than a millisecond for all of the tested instances. The results demonstrate that $\mathbf{LB}_3$ provides the best lower bound among the five lower bounds tested, and that the computation of $\mathbf{LB}_3$ is very fast for all of the tested instances. This suggests that $\mathbf{LB}_3$ might be the best choice of lower-bounding strategy for utilization in the ADP algorithm. Also, the objective values of **LPR** and $\mathbf{LPR}_{\text{cut}}$ show the same behavior as that for the NC instances.

Next, we compare the performance of our upper-bounding heuristics with the heuristics in the literature, the results of which are shown in Table 5. The *%gap* in the table represents the relative gap of the objective value obtained by a heuristic algorithm and a known optimal value (i.e., $\%gap = \frac{(obj-opt)}{opt} \times 100$). From the table, we can see that the AB heuristic provided qualified solutions within a very short period time. In particular, the AB heuristic with $\sigma = 3$ found feasible solutions with a gap of less than 1.3%. On the other hand, the GA tended to yield solutions with a relatively high percentage of gaps in large-sized instances, and the solution time increased rapidly as the problem size increased.

Now, we show the computational performance of the ADP algorithm for the GC instances. $\mathbf{LB}_1$, $\mathbf{LB}_2$, and $\mathbf{LB}_3$ were used to compute the lower bounds at each state in the course of the ADP algorithm. Also, the initial upper bounds were computed using the AB heuristic with $\sigma = 3$, since it returns the best objective value among the heuristic algorithms. Our ADP algorithm was compared with the basic DP algorithm, the results of which are shown in Table 6. Note that the branch-and-bound algorithm of [3] could not be used to solve the GC instances. The computational behavior of the ADP algorithm was very similar to that for the NC instances. Specifically, as expected, ADP3 showed the best performance, assisted by the best lower bound $\mathbf{LB}_3$. The number of states explored in ADP3 also reduced considerably compared with the basic DP algorithm.

5. Conclusion

In this paper, we considered a car resequencing problem (CRP) in automotive paint shops and proposed accelerated dynamic programming (ADP) algorithms to solve it efficiently. Our study contributes to the literature by developing efficient exact and heuristic algorithms that can be applied to both the NC and GC types of problems. In particular, we addressed the GC-type problem and provided extensive computational results for the first time with exact solution methods. The main principle of our algorithms is the incorporation of effective lower- and upper-bounding techniques in the standard dynamic programming framework to reduce the size of the search space and thereby reduce the computational time. Through computational experiments on a wide range of random instances, we showed that the proposed algorithms significantly

Table 5
Comparison of upper-bounding heuristics on GC instances.

Instance			Simple rule	GA [6]		AB ($\sigma = 2$)		AB ($\sigma = 3$)	
config.	*#color*	*dist.*	*%gap*	*%gap*	*time*	*%gap*	*time*	*%gap*	*time*
5×6	10	d_1	35.4	2.7	5.01	1.7	0.03	0.0	0.11
		d_2	29.1	1.9	5.23	1.4	0.04	0.1	0.13
		d_3	28.7	2.0	5.46	0.6	0.04	0.1	0.12
	20	d_1	33.6	1.5	5.04	1.4	0.04	0.1	0.14
		d_2	27.3	1.3	5.31	1.1	0.04	0.0	0.14
		d_3	26.6	1.4	5.92	0.7	0.06	0.0	0.19
7×8	10	d_1	44.1	15.4	13.55	2.7	0.86	0.3	5.97
		d_2	41.2	15.3	14.35	3.0	0.86	0.5	7.44
		d_3	40.9	14.7	15.30	2.6	0.75	0.3	6.27
	20	d_1	36.6	14.2	13.71	4.1	0.93	1.3	7.55
		d_2	38.9	15.5	14.76	3.4	0.97	0.8	9.06
		d_3	33.3	11.7	16.18	2.6	0.75	0.5	6.54
10×3	10	d_1	31.3	7.3	5.58	0.9	1.62	0.0	5.73
		d_2	34.9	9.5	5.99	1.7	1.43	0.0	6.74
		d_3	39.7	7.7	6.20	0.1	0.70	0.0	3.34
	20	d_1	31.5	6.8	5.79	0.0	2.22	0.0	9.80
		d_2	32.7	9.2	6.28	0.4	1.38	0.0	5.75
		d_3	31.5	6.4	6.94	0.2	1.51	0.0	5.75
3×10	10	d_1	24.0	0.2	4.17	1.7	0.01	0.0	0.01
		d_2	21.7	0.3	4.39	0.8	0.01	0.0	0.01
		d_3	18.4	0.1	4.68	0.8	0.01	0.0	0.02
	20	d_1	21.0	0.0	4.27	0.5	0.01	0.0	0.01
		d_2	19.8	0.3	4.49	0.7	0.01	0.1	0.01
		d_3	19.0	0.4	4.89	0.3	0.01	0.0	0.02

Table 6
Performance comparison of exact algorithms on GC instances.

Instance			DP[a]	ADP1[a]		ADP2[a]		ADP3[a]	
config.	*#color*	*dist.*	*time*	*time*	*%state*	*time*	*%state*	*time*	*%state*
5×6	10	d_1	0.16	0.14	75.5	0.09	49.7	0.08	46.7
		d_2	0.18	0.16	73.5	0.12	55.4	0.11	53.5
		d_3	0.21	0.15	64.1	0.12	48.5	0.11	47.6
	20	d_1	0.17	0.17	79.4	0.13	56.4	0.11	52.8
		d_2	0.20	0.18	70.0	0.14	53.4	0.13	52.3
		d_3	0.27	0.18	56.1	0.16	46.9	0.15	46.6
7×8	10	d_1	80.7	82.15	85.6	46.75	50.2	39.98	46.7
		d_2	94.28	96.25	84.7	68.07	63.6	61.64	61.1
		d_3	113.98	103.70	79.2	75.62	60.7	69.48	53.2
	20	d_1	80.02	117.76	80.4	69.53	54.2	55.56	50.8
		d_2	90.12	113.85	72.2	60.26	57.7	61.65	56.8
		d_3	138.15	154.28	54.1	107.85	37.0	101.63	37.0
10×3	10	d_1	30.65	6.24	26.3	3.93	17.0	3.86	16.6
		d_2	40.36	6.36	21.4	4.45	14.9	4.19	14.8
		d_3	47.29	3.06	9.5	2.04	6.4	1.98	6.3
	20	d_1	36.49	9.62	28.8	6.76	20.8	6.35	20.1
		d_2	43.70	4.84	14.1	3.51	10.0	3.17	9.8
		d_3	66.76	4.11	8.2	2.98	6.0	2.79	6.0
3×10	10	d_1	0.01	0.01	91.8	0.01	70.4	0.01	64.7
		d_2	0.01	0.01	88.4	0.01	70.3	0.01	66.4
		d_3	0.01	0.02	90.7	0.02	77.7	0.01	75.4
	20	d_1	0.01	0.01	90.4	0.01	69.1	0.01	62.5
		d_2	0.01	0.02	89.3	0.02	74.4	0.02	72.3
		d_3	0.01	0.02	86.6	0.02	78.0	0.02	77.4

[a] All instances were solved to optimality.

reduce computational effort, outperformed the relevant existing algorithms, and are therefore more practically applicable. Our results show that well-tailored exact algorithms can be more efficient than meta-heuristic algorithms which are usually motivated by the theoretical complexity of $\mathcal{NP}$-hard optimization problems.

The operation of the storage system shown in Fig. 1, as mentioned in Section 1, is performed dynamically. Each unit of time (about 1 minute in a typical plant) a car is retrieved from one of the upstream conveyors and a new car is stored in one of the upstream conveyors. Therefore, the car resequencing achieved by controlling retrieval sequences has an on-line character; the car resequencing is performed in practice by a combination of several prespecified storage and retrieval

policies such as dispatching rules (for example, see Moon et al. [2]). In this dynamic environment, the proposed algorithms can be directly used to setup a retrieval policy to replace the myopic dispatching rules. For example, the process of retrieval can be performed following a preplanned retrieval sequence determined by the proposed algorithms. This retrieval sequence is reoptimized periodically to reflect the updated status of the upstream conveyors. For example, based on the results shown in Table 5, the proposed algorithms can achieve almost the lowest possible changeover costs, while a simple rule used in practice incurs at least 30% higher changeover costs for the 7×8 configuration. This means that the proposed algorithms are effective options to better manage the multi-million-dollar changeover costs.

The sequence of cars stored in each upstream conveyor can significantly affect the overall changeover costs in the downstream conveyor. This implies that the changeover costs can be further reduced by better controlling the storage operations. Therefore, it is worthwhile to study the car resequencing problem by considering not only retrieving the car from the storage system but also selecting the appropriate upstream conveyor to store the cars from the body shop.

Acknowledgment

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A1A01057719). The authors would like to thank the authors of [6] for kindly providing us with their GA-based heuristic code for our computational tests. The authors also would like to thank the editors and anonymous referees for their constructive and helpful comments.

References

[1] D.L. Myron, T.L. Magnanti, Paint blocking in ford's in-line vehicle sequencing environment. leaders for manufacturing program, in: Technical Report, Cambridge, 1996.
[2] D.H. Moon, H.S. Kim, C. Song, A simulation study for implementing color rescheduling storage in an automotive factory, Simulation 81 (2005) 625–635.
[3] S. Spieckermann, K. Gutenschwager, S. Voß, A sequential ordering problem in automotive paint shops, Int. J. Prod. Res. 42 (2004) 1865–1878.
[4] T. Epping, W. Hochstattler, P. Oertel, Some results on a paint shop problem for words, Electron. Notes Discret. Math. 8 (2001) 31–33.
[5] T. Epping, W. Hochstättler, P. Oertel, Complexity results on a paint shop problem, Discret. Appl. Math. 136 (2004) 217–226.
[6] S.S. Ko, Y.H. Han, J.Y. Choi, Paint batching problem on m-to-1 conveyor systems, Compute. Oper. Res. 74 (2016) 118–126.
[7] F.Y. Ding, H. Sun, Sequence alteration and restoration related to sequenced parts delivery on an automobile mixed-model assembly line with multiple departments, Int. J. Prod. Res. 42 (8) (2004) 1525–1543.
[8] S.A. Hartmann, T.A. Runkler, Online optimization of a color sorting assembly buffer using ant colony optimization, in: Proceedings of the Operations Research 2007, Springer, 2008, pp. 415–420.
[9] H. Sun, S. Fan, X. Shao, J. Zhou, A colour-batching problem using selectivity banks in automobile paint shops, Int. J. Prod. Res. 53 (2015) 1124–1142.
[10] M. Lermen, K. Reinert, The practical use of the A* algorithm for exact multiple sequence alignment, J. Comput. Biol. 7 (2000) 655–671.
[11] F. Ahmadizar, P. Shahmaleki, Group-shop scheduling with sequence-dependent set-up and transportation times, Appl. Math. Model. 38 (2014) 5080–5091.
[12] M. Ebrahimi, S.M.T.F. Ghomi, B. Karimi, Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates, Appl. Math. Model. 38 (2014) 2490–2504.
[13] T. Keshavarz, M. Savelsbergh, N. Salmasi, A branch-and-bound algorithm for the single machine sequence-dependent group scheduling problem with earliness and tardiness penalties, Appl. Math. Model. 39 (2015) 6410–6424.
[14] M. Conforti, G. Cornuéjols, G. Zambelli, Integer Programming, Volume 271 of Graduate Texts in Mathematics, Springer, Berlin, 2014.
[15] T. Epping, W. Hochstättler, Sorting with line storage systems, in: Proceedings of the Operations Research 2002, Springer, 2003, pp. 235–240.
[16] Xpress, Xpress 8.0, 2016, http://www.fico.com/en.