A Hybrid Genetic Algorithm for Generalized Order Acceptance and Scheduling

Sasan Mahmoudinazlou[1] and Hadi Charkhgard[2]

[1]Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL 33620, sasanm@usf.edu

[2]Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL 33620, hcharkhgard@usf.edu

Abstract

In this paper, a novel approach is presented to address a challenging optimization problem known as Generalized Order Acceptance Scheduling. This problem involves scheduling a set of orders on a single machine with release dates, due dates, deadlines, and sequence-dependent setup times judiciously to maximize revenue. In view of resource constraints, not all orders can be accommodated; accordingly, a careful selection and sequencing process is required. Our proposed method leverages a hybrid genetic algorithm in conjunction with approximate dynamic programming to address sequencing and acceptance decisions, respectively. The algorithm's performance is enhanced through custom-built local searches, guided by order-specific weights that inform acceptance and rejection choices. Additionally, a rank-based representation is used to quantify the differences between individuals and to promote diversity among the population. Numerical evaluations conducted on a well-established benchmark dataset demonstrate the effectiveness of our approach. The results of our comparative analysis against six baseline algorithms from the literature indicate that our method has the potential to significantly improve the outcome of order acceptance scheduling.

Keywords: Order Acceptance, Scheduling, Genetic Algorithm, Dynamic Programming

1 Introduction

Order Acceptance Scheduling (OAS) is a combinatorial optimization problem focused on efficiently managing orders to maximize overall revenue. In its simplest form, this problem revolves around a single machine tasked with handling numerous orders, each characterized by different processing times, deadlines, and revenue impacts. To optimize total revenue, the OAS problem involves the selection of a subset of orders from a given pool and the organization of their processing on a single machine. The complexity arises from the constraints imposed by factors such as release times, processing times, setup times, due dates, and deadlines for each order. The interplay of these constraints creates a complex decision space where the objective is to make judicious choices in order selection and sequencing to attain the highest possible revenue.

The significance of OAS lies in its extensive applicability across various sectors. For instance, in manufacturing, OAS plays a pivotal role, as this sector heavily relies on the timely processing of orders to meet customer expectations and sustain competitiveness. Similarly, OAS is crucial in supply chain management, where the timing and selection of orders are instrumental in maintaining optimal inventory levels and ensuring on-time deliveries. Moreover, it is worth noting that the OAS problem extends beyond industrial sectors and finds relevance in service sectors. For example, in healthcare, optimizing the allocation of medical resources is essential to achieve the best possible outcomes for patients.

The body of literature on OAS and its variations is extensive. For instance, Esmaeilbeigi et al. (2016) introduced new Mixed Integer Linear Programming (MILP) formulations for OAS along with valid inequalities. Additionally, de Weerdt et al. (2021) proposed an exact algorithm based on dynamic programming to address OAS, and Wang et al. (2015) extended OAS to two parallel machines, offering two heuristics and a Lagrangian relaxation-based exact solution methods. However, the primary focus of this paper is on a particular variant of OAS, which has received relatively less attention in the OAS literature due to its complexity. This variation was first introduced by Og et al. (2010), to the best of our knowledge, and it involves sequence-dependent setup times. Throughout this paper, we will use the term Generalized Order Acceptance Scheduling (GOAS) to refer to this variant. To provide a better context for our contributions, a brief literature review on GOAS is provided next.

1.1 Literature Review

A Mixed Integer Linear Programming (MILP) approach is used by Og et al. (2010) to formulate the GOAS problem, which is capable of providing optimal solutions for up to 15 orders. The authors propose three heuristic algorithms for solving larger instances. The study also generates a benchmark set of instances that has been used in most of the subsequent studies. A variety of algorithms have been proposed to solve the GOAS problem, both exact and heuristic. There is a branch-and-bound algorithm proposed by Nobibon and Leus (2011) that is capable of solving instances with a size up to 50 orders to optimality. An arc-time-indexed mathematical formulation and two exact algorithms based on it are proposed by Silva et al. (2018), including a Lagrangian relaxation method and a column generation algorithm.

In the literature, heuristic algorithms have been used more frequently for the solution of GOAS problems. A Tabu Search is proposed by Cesaret et al. (2012) for solving the problem. A solution is represented by a vector in which the i^{th} entry indicates the position of the order o_i in the sequence if accepted, whereas zero if not accepted. Lin and Ying (2013) use an Artificial Bee Colony algorithm, in which a solution is represented by a linear permutation of orders. A similar representation of the solution is offered by Chaurasia and Singh (2017) in two evolutionary algorithms. In both of these papers, orders that violate the deadline constraint are rejected in a simple manner. Artificial bee colonies-based heuristics are also proposed by Wang et al. (2013) for a variation of a GOAS involving two machines. A Genetic algorithm for GOAS based on tree representations have been

proposed by both Park et al. (2013) and Nguyen (2016). A genetic algorithm based on dispatching rules is proposed by Nguyen et al. (2015), in which each individual is divided into T segments and T dispatching rules are selected from a set of candidate rules. A biased random-key genetic algorithm was developed by He et al. (2019) in which each chromosome is represented by a vector of numbers between 0 and 1. For decoding, the orders are sorted according to their gene values and those not in violation of the deadline are scheduled. By using an ALNS algorithm, they improve GA's solution further. Mahmoudinazlou et al. (2023) hybridizes an Imperial Competitive Algorithm (ICA) with Simulated Annealing (SA). ICA is a fairly new population-based metaheuristic algorithm. To solve GOAS, Tarhan and Oğuz (2022) employ a combination of metaheuristic algorithms and mathematical programming. The proposed algorithm is comprised of a time-bucket MILP model, a variable neighborhood search algorithm, and a tabu search algorithm.

1.2 Contributions

To solve GOAS problem, two decisions must be made: first, selecting some tasks out of the available orders, and second, scheduling them in an order to maximize revenue. To achive this, a hybrid genetic algorithm is proposed in this study, where the GA is responsible for sequence-level decisions and an approximate dynamic programming algorithm is used to deal with acceptance-level decisions. This approach was motivated by our previous work (Mahmoudinazlou and Kwon, 2024), where a divide and conquer approach was employed to address the Multiple Traveling Salesman Problem. In that study, the GA places the cities in order, whereas the DP splits the sequence into m different tours. Additionally, in this study, some neighborhood searches are performed to facilitate faster convergence to (near) optimal solutions. Among the major contributions of this paper are:

1. A permutation-based representation is used to encode the solution, which includes all the orders. To handle the acceptance/rejection decisions for the given sequence, an approximate dynamic programming approach is applied, which improves the efficiency of the decision-making process.

2. A new crossover is designed that takes into account both sequence-related and acceptance-related decisions.

3. To enable smart removal and addition of orders to the solution, local searches are performed by assigning a weight to each job.

4. By using a rank-based representation, the differences between individuals in the population are quantified in order to increase the diversity of the population.

5. As part of our evaluation, we tested our algorithm against a well-known benchmark set and compared it to six baseline algorithms. The results show that our algorithm outperforms all six baseline algorithms.

Below is an overview of the paper's organization. In section 2 the problem is formally described. The details of our proposed method are outlined in Section 3. The results of the computational experiments are presented in section 4, which compares our results with those of the baseline algorithms. Lastly, we conclude the paper in Section 5 and make suggestions for future research.

2 Problem Description

In GOAS, there are n orders that can be processed on a single machine. Orders are assigned a specified timeframe within which they can be completed. It may be impossible to accomplish all tasks within the specified time frame due to a lack of resources. Therefore, in order to maximize the total revenue, some tasks must be selected and scheduled. Let $\boldsymbol{O} = \{o_1, o_2, ..., o_n\}$ be collection of all available orders. The release time and processing time of order o_i are represented by r_i and p_i respectively. The setup time between order o_i and order o_j if scheduled consecutively, is shown by S_{ij}. In case o_j is the first scheduled order, the setup time of the machine is S_{0j}. The processing of a particular order o_i is only possible after it has been released. Once the order has been released and is ready for scheduling, the machine will be set up accordingly. It is important to note that even the setup process cannot be initiated prior to r_i. The order will be processed within p_i time units once the machine has been set up. Let us assume that order o_j occurs immediately after order o_i. Let C_j be the completion time for the order o_j. Therefore, $C_j = \max(C_i, r_j) + S_{ij} + p_j$ is the completion time for order o_j.

The due date and deadline of order o_i are represented by d_i and $\bar{d}_i$ respectively. The tardiness of order o_i can be calculated as $l_i = \max(0, d_i - C_i)$ If order o_i is completed prior to or on the due date, revenue e_i is earned. The revenue will decrease proportionally to the amount of lateness if the order is completed after the due date but before the deadline. The order will not generate revenue if the task is completed after the deadline. Let w_i be tardiness cost of order o_i where $w_i = \dfrac{e_i}{\bar{d}_i - d_i}$. Therefore, the reduction in revenue between d_i and $\bar{d}_i$ is linear. Consequently, order o_i would generate revenue of $\max(0, e_i - w_i \times l_i)$. The optimal solution of GOAS is a subset of $\{o_1, o_2, ..., o_n\}$ scheduled on machine in a sequence that maximizes the total gained revenue.

3 A Hybrid Genetic Algorithm

The purpose of this section is to describe the structure and details of our proposed Hybrid Genetic Algorithm (HGA). It is best to begin by discussing the representation of the chromosomes and their evaluation mechanism. The chromosome representation in our GA is a sequence of all orders. For instance, $[3, 2, 5, 4, 1]$ is a chromosome representation for a problem with size $n = 5$. However, due to the nature of the orders, which have specific release times and deadlines, and the fact that there is only one machine to process them, not all of the orders can be successfully completed in the optimal solution. This requires a decision to be made regarding the acceptance or rejection of each order. As a naive approach, one would accept as many orders in the sequence as the machine

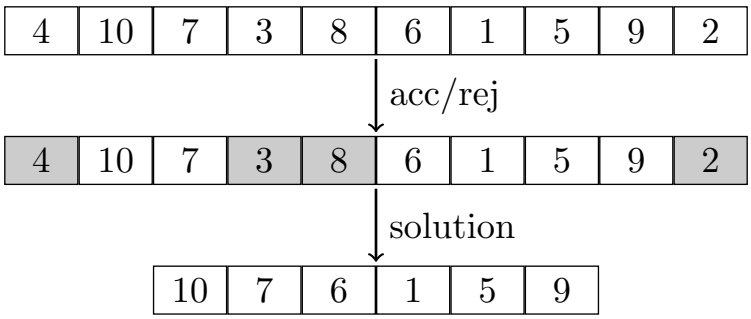

Figure 1: The mechanism of acceptance/rejection decision making by DP (or ADP).

resources would allow. For the population to thrive quickly, however, a more efficient approach is necessary.

3.1 Approximate Dynamic Programming

In order to evaluate a chromosome, acceptance/rejection decisions must be made without changing the sequence of orders. A dynamic programming approach may be designed to make these decisions in an optimal manner. The DP will take a sequence of orders and select the best possible subset of those orders without changing their sequence (Figure 1). We define $\alpha(k,t)$ as the subproblem of optimally deciding acceptance and rejection decisions for orders $o_1, ..., o_k$ up to time t. Let $F(k,t)$ represent the optimal reward for subproblem $\alpha(k,t)$. By using Bellman optimality equations in forward propagation and starting with $F(0,0) = 0$, we aim to find $F(n,t)$ for all $t \leq \max(\bar{\boldsymbol{d}})$, where $\bar{\boldsymbol{d}}$ is the vector that represents the deadlines. We can obtain the optimal solution of DP by using $\max_t(F(n,t))$. Assuming $k_2 > k_1$ and $t' > t$, the Bellman optimality equations are $F(k_2,t') = \max_{k_1,t} \big(F(k_1,t) + R(k_1,k_2,t,t')\big)$, where $R(k_1,k_2,t,t')$ is the revenue obtained by scheduling the order in position k_2 right after the order in position k_1, starting from somewhere after t so that it ends at time t'. If such scheduling is not possible, then $R(k_1,k_2,t,t') = 0$.

The proposed dynamic programming approach will produce a unique solution for any sequence of orders. In this way, we can be assured that at least one sequence of orders will provide the optimal solution to the GOAS problem. It is, however, not possible to use the proposed DP at large scales due to its high computational cost. The proposed dynamic programming has a time complexity of $O(n^2 \times \max(\bar{\boldsymbol{d}}))$. In cases where $n = 100$ orders are involved and $\max(\bar{\boldsymbol{d}}) > 1000$, using DP for chromosome evaluation would be illogical.

In order to overcome this issue, we propose an approximate dynamic programming method, which takes a sequence of jobs and determines the near-optimal acceptance/rejection for the given sequence. This is achieved by removing the time dimension from the proposed DP. Although the output of ADP might not be the optimal acceptance/rejection decision, it produces much better solutions than the naive policy. A detailed discussion of approximate dynamic programming can be found in Algorithm 1. Let $\boldsymbol{O} = \{o_1, o_2, ..., o_n\}$ represent the sequence of orders. If we divide the problem into maximizing the revenue for acceptance/rejection decisions of a subsequence of

$\boldsymbol{O}$ from o_1 to o_k, then Bellman's optimality equation can be used for forward propagation. Let F_k represent the near-optimal revenue from making acceptance/rejection decisions up to order o_k. Assuming $k_2 > k_1$, we can use forward propagation to obtain F_{k_2} from F_{k_1} by equation $F_{k_2} = \max_{k_1}(F_{k_1} + R(k_1, k_2))$, where $R(k_1, k_2)$ is the revenue of scheduling order o_{k_2} after order o_{k_1}, and is calculated as `revenue` in Lines 7-9. Accordingly, calculating F_n leads to the final solution. T_k is used to keep track of completion times, whereas P_k is used to extract the final solution. An order's predecessor level is represented by P_k.

Algorithm 1 Approximate Dynamic Programming

1: $F_k \leftarrow 0 \quad \forall k = 0$ to n
2: $T_k \leftarrow 0 \quad \forall k = 1$ to n
3: $P_k \leftarrow 0 \quad \forall k = 1$ to n
4: $o_0 = 0$
5: **for** $k_1 = 0$ to $n - 1$ **do**
6: **for** $k_2 = k_1 + 1$ to n **do**
7: $t = \max(T_{k_1}, r[o_{k_2}]) + S_{o_{k_1}, o_{k_2}} + p[o_{k_2}]$
8: `tardiness` $= \max(0, t - d[o_{k_2}])$
9: `revenue` $= \max(0, e[o_{k_2}] -$ `tardiness` $* w[o_{k_2}])$
10: **if** $F_{k_1} +$ `revenue` $> F_{k_2}$ **then**
11: $F_{k_2} \leftarrow F_{k_1} +$ `revenue`
12: $T_{k_2} \leftarrow t$
13: $P_{k_2} \leftarrow k_1$
14: **end if**
15: **end for**
16: **end for**
17: return $\max_{k=1}^{n}(F_k)$

By using the approximate dynamic programming for a given sequence of orders, the near-optimal acceptance/rejection decisions can be determined in $O(n^2)$, where n is the total number of orders.

According to our experiments, approximate DP provided a better trade-off between computational time and solution quality.

3.2 GA Structure

A detailed description of the HGA is provided in Algorithm 2. As a starting point, heuristic methods are used in order to generate μ individuals (line 1). Each of the following steps is repeated until the stopping condition is met (lines 2-15). The algorithm terminates when there is no improvement after It_{STOP} iterations or t_{STOP} seconds, whichever occurs first. In the first step, the population is sorted according to fitness that is calculated based on the revenue and a diversification multiplier (line 3). The diversification method will be discussed in more detail in Section 3.6. Two individuals will then be selected as parents from the population (line 4). In this study, *Tournament Selection* was used to select the parents. In tournament selection, a subset of individuals is selected at

random from a population. In this paper, $k_{\text{TOURNAMENT}}$ represents the number of individuals within the subset, which is a user-defined parameter. Each individual within this subset competes against the other, and the one with the highest fitness value is selected as a parent. This process is repeated for selecting the second parent. Using crossover with two parents will result in one child (line 5) that is subjected to the ADP algorithm for fitness evaluation (line 6). Next, neighborhood search functions will be applied to the new offspring in order to improve it (line 7). Once the population reaches the size of $\mu+\lambda$, a survival plan is implemented. The population is sorted based on objective function, and the best μ individuals are maintained and the rest are discarded (line 9-11). As a further effort to help the algorithm avoid local optima, the population will be diversified when no improvement is observed after It_{DIV} iterations. This step consists of keeping n_{BEST} top individuals, discarding the rest, and generating new individuals until the population size returns to μ. Creating new individuals follows the same process as generating the initial population (lines 12-14).

Algorithm 2 Hybrid Genetic Algorithm

1: $\Omega =$ `initial_population()` ▷ Section 3.3, Algorithm 3
2: **while** Stopping condition is not met **do**
3: `sort`(Ω) ▷ Based on fitness and diversification factor
4: Select ω_1 and ω_2 from Ω
5: $\omega \leftarrow$ `crossover`(ω_1, ω_2) ▷ Section 3.4
6: `evaluate`(ω) ▷ Approximate Dynamic Programming, Algorithm 1
7: `neighborhood search`(ω) ▷ Section 3.5
8: $\Omega_{\text{F}} \leftarrow \Omega_{\text{F}} \cup \{\omega\}$
9: **if** `size`$(\Omega) = \mu + \lambda$ **then**
10: `select_survivors`(Ω)
11: **end if**
12: **if** `best`(Ω) not improved for It_{DIV} iterations **then**
13: `diversify`(Ω)
14: **end if**
15: **end while**
16: Return `best`(Ω)

The remainder of this section is summarized as follows: Section 3.3 describes the process of generating the initial population. Section 3.4 provides an explanation of two crossover functions developed in this study. Then, we explain in section 3.5 the neighborhoods that are used in local search to improve the generated offspring. Finally, the

3.3 Initial population

Based on our chromosome representation and evaluation, any permutation of orders can lead to a feasible solution if ADP is applied to it. In evolutionary algorithms, however, the quality of the initial population plays a very important role in determining the speed of convergence. Therefore, it is much more beneficial to begin with solutions that are superior to random solutions. Algorithm 3 describes the process for generating initial solutions in this study. First, the orders are sorted

according to their release date. The available orders are listed at each time point and based on probabilities proportional to $RTR = \frac{\texttt{revenue}}{\texttt{time on machine}}$, an order is selected. An order's ratio increases if revenue is higher and time spent on the job is shorter. In this way, orders that produce more revenue in less time have a greater chance of being scheduled. In addition, since sampling is being used, each time the solution will be different. Algorithm 3 is used to generate the initial population and to diversify when necessary.

Algorithm 3 Generating initial greedy solution

$t \leftarrow 0$
$S^* \leftarrow \emptyset$
$R \leftarrow \{o_1, o_2, ..., o_n\}$ ▷ List of all orders
$l \leftarrow 0$ ▷ Last scheduled order
repeat
  $J = \{o_i \in R | r[o_i] <= t\}$ ▷ All available orders at time t
  if $J = \emptyset$ **then**
    $t \leftarrow \min_{o_i \in R}(r[o_i])$
  else
    for $o_i \in J$ **do**
      $C_{o_i} = t + S_{l,o_i} + p[o_i]$ ▷ Completion time
      $\texttt{tardiness} = \max(0, C_{o_i} - d[o_i])$
      $\texttt{revenue} = \max(0, e[o_i] - \texttt{tardiness} * w[o_i])$
      $RTR_{o_i} = \frac{\texttt{revenue}}{C_{o_i} - t}$ ▷ Revenue to time ratio
    end for
    $l = \texttt{sample}(J)$ proportional to RTR_{o_i}
    $S^* \leftarrow S^* \cup l$
  end if
until There is no job in R that can be scheduled with positive revenue
Return S^* ▷ One initial solution

3.4 Crossover

A total of three crossovers are used in our GA in order to achieve a greater degree of diversity. The representation of the chromosomes in our study allows the use of any permutation-based crossover. The *Order Crossover* (OX1) (Davis et al., 1985) is selected from the literature to be used in this study. OX1 creates two crossover points randomly in the parent, then copies the segment between the crossover points to the offspring. Next, the remaining unused numbers from the second parent are copied to the child, in the same order in which they appear in the second parent.

As OX1 is a crossover over the permutations, the decision as to whether to accept or reject orders will be solely based on ADP. Therefore, two additional crossovers have been developed that are more aligned with the specifics of the problem. The first one is a new crossover that inherits the acceptance/rejection decision from one parent and the sequence decision from the other parent. The term *Acceptance Sequence Crossover* (ASX) was coined for this purpose (Figure 2). The first

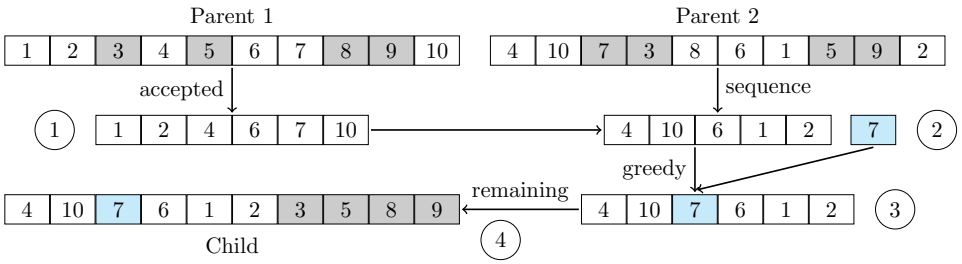

Figure 2: An example of ASX crossover. The highlighted genes in parents represent the rejected orders.

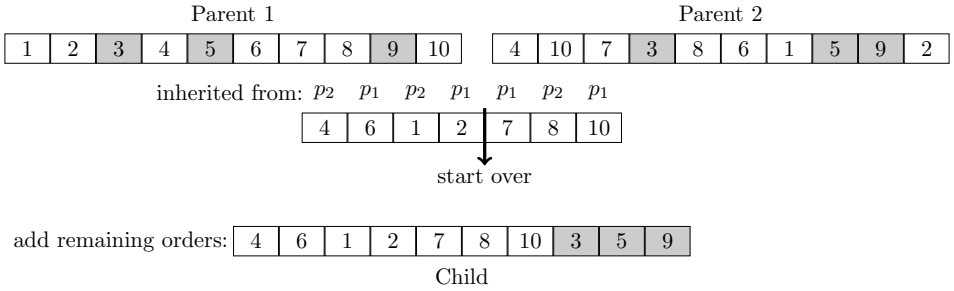

Figure 3: An example of SSX crossover. The highlighted genes in parents represent the rejected orders.

parent's accepted jobs are taken (step 1), then they are arranged in accordance with the order in which they appear in the second parent's solution (step 2). Furthermore, jobs that are accepted in the first parent but not in the second, are inserted into the sequence greedily, i.e., we try each of the jobs in all positions and insert them where it yields the highest revenue (step 3). It is necessary to add all the unaccepted jobs to the end of the child in order to preserve its full size (step 4).

The second new crossover is designed to account for the sequence-dependent setup time characteristic (Figure 3). A random parent is selected, and the first job in its solution is added to the child. Then, the following process is repeated: let o_j be the last scheduled job on the child. Using the other parent, we find o_j and add the job immediately after it. In the event that this process is unsuccessful in finding a new job, we revert to step one and start over. The reason may be that o_j is not present in the solution of the other parent, or it is the last job in the solution, or it was added previously. The remaining jobs must also be added at the end of the child (last step). A *Sequence to Sequence Crossover* (SSX) is what we call this crossover. The use of these three crossovers allows OX1 to diversify the population in terms of sequence, while ASX and SSX add diversity in terms of acceptance and sequence.

3.5 Neighborhood search

Our hybrid method employs local searches in order to facilitate faster convergence. Following the crossover, we employ local searches to improve the quality of the generated offspring. There is, however, a potential trade-off here. A population can be trapped in local optima if too much local

search is performed. Therefore, we attempt to improve the offspring with a probability that we set equal to τ. During our experiments, we observed that tight due dates necessitated more local searches. In this study, five different neighborhoods are taken into account, as shown in Figure 4. S represents the sequence of acceptable jobs for the individual, while S' represents the list of rejected jobs. The following is a detailed description of each neighborhood.

- N_1: Selects a job from S' and places it in S.
- N_2: Selects a job from S' and places it between two consecutive jobs in S, while swapping them.
- N_3: Replaces a job from S with a job from S'.
- N_4: Removes a job from S while selecting a job from S' and placing it in S.
- N_5: Selects two jobs from S', replaces one with a job in S while adding the other to S

It can be seen that all of the neighborhoods are primarily related to the acceptance level of the decision. Therefore, sequence-level decisions are left to crossover functions in GA. In order to improve the efficiency of the local search process, two ideas have been implemented to make the search smarter. First, neighborhood selections are made according to probabilities which are updated through a roulette wheel mechanism. Thus, the algorithm will employ more beneficial neighborhoods according to the specific instance. Secondly, the selection of jobs from S and S' is not subject to pure randomness. Two types of weights are introduced for orders, static weights and dynamic weights. An order's static weight is equal to $\frac{e[o_j]}{p[o_j] + S_{avg,o_j} + S_{o_j,avg}}$, which remains constant throughout the algorithm. If an order o_j is accepted, the dynamic weight is $\frac{\texttt{revenue}[o_j]}{\texttt{time on machine}[o_j]}$, same as the RTR proposed in Section 3.3. Adding a job from S' to S is done by random sampling with probabilities proportional to static weights. Consequently, we increase the chances of jobs that lead to higher revenues within a shorter period of time. Similarly, we use probabilities proportional to the inverse of dynamic weights when removing a job from S.

Further, we select the location of the new job that is to be added to S only from positions that are compatible with the deadline. The starting and completion times of all jobs are pre-recorded for this purpose.

3.6 Diversification

Generally, genetic algorithms tend to produce populations that are similar to the best individual, especially when populations are small. This may lead to the algorithm being trapped in local optima. The chances of reaching global optima are higher for a diverse population because it has more exploration opportunities. The process of mutation is one of the methods employed by GAs to diversify their populations. We have however not used any mutations in this study due to the fact that our experiments showed that they were not very effective in our method. In this study,

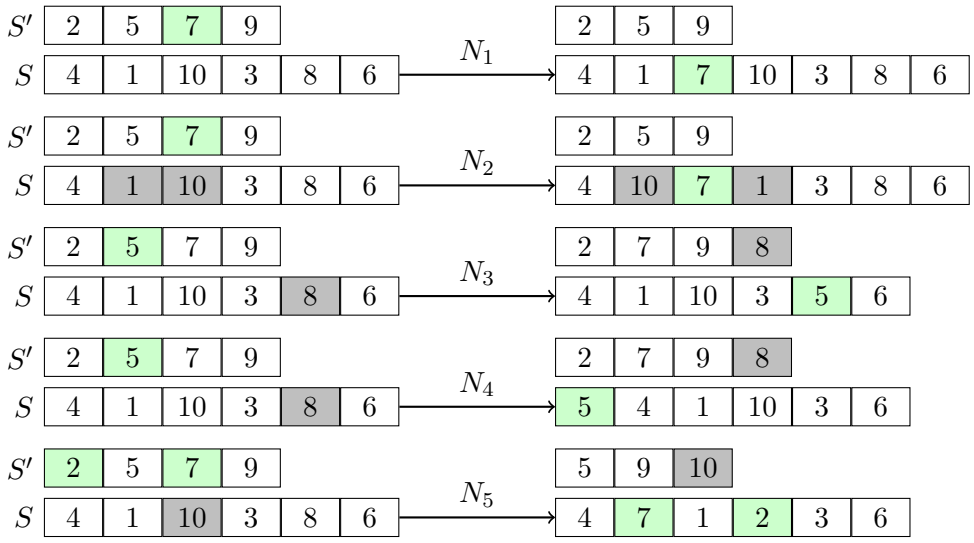

Figure 4: Illustration of local search functions ($L_1 - L_8$)

diversifying the population is accomplished in two ways. We do not allow the coexistence of two individuals with identical solutions in our GA.

To further diversify the population, we evaluate the chromosome's fitness based on the revenue its solution generates, as well as its distance from its adjacent chromosomes after sorting by fitness. In order to quantify the distance between the two individuals, we first convert the OAS solution into a rank-based representation. The rank-based representation of each individual consists of an array with size n, which represents the position of all jobs in the solution. A value will be assigned to each job in this representation, which represents its position within the solution. The corresponding value will be zero if a job is not accepted. Consider $P = [4, 3, 8, 5, 10, 7]$ as the solution of a chromosome when $n = 10$. Therefore, the rank-based representation of this chromosome would be $R = [0, 0, 2, 1, 4, 0, 6, 3, 0, 5]$. A distance between two individuals P_1, P_2 with rank-based representations R_1, R_2 can be calculated as follows:

$$\delta(P_1, P_2) = \frac{1}{n} \sum_{i=1}^{n} \frac{|R_1[i] - R_2[i]|}{M_r},$$

where M_r is the maximum rank value in R_1 and R_2.

The distance between two representations is measured between zero and one, where distances close to zero indicate similar representations and distances close to one indicate differences. Diversity is quantified by calculating the diversity contribution $\Delta(P)$ based on the average distance between an individual P and two of its closest neighbors. In order to calculate the fitness function of each individual, the following formula is used:

$$\texttt{fitness}(P) = \texttt{revenue}(P) \times \gamma^{\Delta(P)},$$

where `revenue`(P) is calculated using ADP and γ is the diversification factor, a hyperparameter of the algorithm.

4 Computational results

In order to test the performance of our algorithm, we used a well-studied benchmark set of instances generated by Cesaret et al. (2012). Data set instances range in size from $10, 20, 15, 25, 50$ and 100. However, we report results only for instances greater than or equal to 25. Each problem size has been categorized with different tardiness factors τ and due date ranges R. These factors have an impact on the generation of release times, due dates, as well as deadlines for orders. A detailed description of the generation of instances can be found in Cesaret et al. (2012). Five values of $0.1, 0.3, 0.5, 0.7$ and 0.9 are considered for τ and R. Ten samples are generated for each combination, so there are 250 samples per size.

The baseline algorithms used for comparison are TS (Cesaret et al., 2012), ABC (Lin and Ying, 2013), DRGA (Nguyen et al., 2015), GA, LOS (Nguyen, 2016) and HSSGA (Chaurasia and Singh, 2017). All algorithms solve each instance ten times and all reports are based on average performance over 10 runs including our algorithm.

4.1 Parameter settings

The algorithm is implemented using Julia programming language on a Mac computer with 16 GB of RAM and an Apple M1 processor. The parameters used in our HGA are $\mu = 20$, $\lambda = 10$, $k_{\text{TOURNAMENT}} = 2$, $It_{\text{DIV}} = 1500$, $n_{\text{BEST}} = 0.2\mu$, $\gamma = 1.2$, $It_{\text{STOP}} = 50000$ and t_{STOP} is set to $5, 30$ and 60 seconds for $n = 25, 50$ and 100 respectively. Therefore, the computational times for all of the instances with size $n = 25, 50$ and $n = 100$ are less than $5, 30$ and 60 seconds respectively.

4.2 Analysis of results

As mentioned earlier, for each combination of n, τ and R, there exist 10 instances in the dataset. While reporting the results on this benchmark set, all studies use columns named Min, Avg., and Max. The numbers in the tables represent the gap between the results of a method and the upper bound obtained from MILP Cesaret et al. (2012). The gap for each instance i is calculated using the following formula:

$$\text{GAP}_i = \frac{\text{UB}_i - \text{LB}_i^{heuristic}}{\text{UB}_i} \times 100\%$$

We have provided detailed comparisons of results between our HGA and the baseline algorithms in Tables 1, 2 and 3. The Min, Avg., and Max columns represent the minimum, average, and maximum gaps over 10 instances of each (n, τ, R) combination. With the exception of HSSGA, all baseline algorithms have rounded their gaps to integers. The method of rounding was not discernible in these studies as to whether it was rounding to the nearest integer or rounding down. Therefore, we follow Chaurasia and Singh (2017)'s method of reporting to two decimal places.

		TS			ABC			DRGA			GA			LOS			HSSGA			HGA		
τ	R	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
0.1	0.1	1	4	6	1	3	4	2	3	4	1	2	3	1	2	3	1.09	2.40	3.16	0.76	1.86	2.74
	0.3	2	3	6	1	2	6	1	2	6	1	2	5	0	2	5	0.69	1.88	5.78	0.69	1.56	2.86
	0.5	1	2	4	0	1	2	1	1	3	0	1	2	0	1	3	0.00	0.86	1.81	0.00	0.73	1.45
	0.7	0	1	4	0	1	3	0	1	3	0	0	2	0	0	2	0.00	0.39	1.67	0.00	0.36	1.38
	0.9	0	1	2	0	0	2	0	1	2	0	0	2	0	0	2	0.00	0.35	2.09	0.00	0.24	1.74
0.3	0.1	2	4	5	1	3	5	2	3	5	1	3	4	1	3	4	1.21	2.53	4.55	0.89	2.37	3.99
	0.3	3	5	7	1	3	6	2	3	5	1	3	4	2	3	5	1.12	3.13	5.57	1.72	3.02	4.42
	0.5	2	3	6	2	2	5	2	2	3	0	2	3	0	2	3	1.12	1.80	3.50	0.89	1.68	3.50
	0.7	1	2	6	1	2	5	1	2	5	0	1	5	0	1	5	0.00	1.61	4.61	0.00	1.47	4.62
	0.9	0	2	4	0	1	3	0	1	3	0	1	2	0	1	2	0.00	1.32	3.38	0.00	0.85	2.50
0.5	0.1	3	6	7	3	5	7	2	4	7	1	4	6	1	4	5	1.68	4.01	6.76	1.68	3.88	6.45
	0.3	3	5	9	3	4	7	2	5	8	2	4	7	1	4	7	1.68	4.19	7.29	1.72	3.92	7.15
	0.5	2	5	8	2	4	6	1	5	6	1	4	6	1	4	6	0.97	4.14	6.37	0.97	4.15	6.05
	0.7	2	6	11	0	3	12	2	4	7	0	4	7	0	4	7	0.64	4.37	7.47	0.64	4.17	7.17
	0.9	1	4	7	1	3	7	1	3	7	1	3	7	1	3	6	1.03	3.05	6.79	0.68	2.99	6.79
0.7	0.1	3	9	18	1	8	16	1	8	16	0	7	14	0	7	15	0.93	7.48	14.86	0.93	7.43	15.64
	0.3	7	10	14	5	9	12	5	9	13	5	8	12	5	8	12	5.78	8.61	12.50	5.05	8.32	12.40
	0.5	7	12	15	5	10	14	6	10	14	5	10	14	5	10	14	6.61	10.43	14.03	5.37	10.00	14.03
	0.7	2	8	14	2	7	12	2	7	12	1	6	12	1	6	12	1.69	6.65	12.08	1.69	6.49	12.08
	0.9	3	10	15	0	8	14	1	8	14	0	8	13	0	8	13	0.01	8.07	13.64	0.01	8.07	13.64
0.9	0.1	0	1	6	0	1	5	0	1	5	0	1	5	0	1	5	0.00	0.54	5.42	0.00	0.54	5.42
	0.3	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0.00	0.00	0.01	0.00	0.00	0.01
	0.5	0	4	12	0	3	12	0	2	12	0	2	12	0	2	12	0.00	2.49	12.30	0.00	2.49	12.30
	0.7	0	8	25	0	7	21	1	7	21	0	6	21	0	6	21	0.00	6.76	20.99	0.00	6.62	20.99
	0.9	0	7	22	0	6	19	1	6	20	0	6	19	0	6	19	0.00	6.08	19.33	0.00	5.99	19.10
	Avg.	2	5	9	1	4	8	1	4	8	1	4	7	1	4	8	1.05	3.73	7.84	0.95	3.57	7.54

Table 1: Performance of HGA compared to baseline algorithms for $n = 25$

As can be seen in Table 1, almost all algorithms have similar performance for instances with size $n = 25$. A detailed comparison of ABC, DRGA, GA, and LOS is difficult due to rounding errors. HGA, however, performs better in Min, Avg., and Max gaps when compared to HSSGA. The HGA clearly outperforms the baseline algorithms on all measures for instances with $n = 50$ and $n = 100$ according to Tables 2 and 3.

5 Conclusions

A hybrid genetic algorithm was developed in this study to address the complex optimization challenge presented by the generalized order acceptance problem in the scheduling domain. Our approach carefully combines genetic algorithms with approximate dynamic programming in order to ensure proficient handling of both sequencing and acceptance decisions. Incorporating intelligent local searches, which are driven by order-specific weightings, enhances convergence while using a rank-based representation fosters population diversity. The empirical evaluations of our proposed method, conducted on a widely recognized benchmark dataset, provide compelling evidence of its effectiveness.

The findings of this study also underscore the importance of using a combination of heuristics and exact algorithms. There is a great deal of value in heuristics, which are prized for their speed, and they complement the precision of exact algorithms that ensure superior outcomes. As a result of this amalgamation, we are able to benefit from the strengths of both approaches. By refining

		TS			ABC			DRGA			GA			LOS			HSSGA			HGA		
τ	R	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
0.1	0.1	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1.03	1.46	2.17	0.69	0.96	1.56
	0.3	1	2	4	1	2	2	1	2	3	1	2	4	1	2	3	0.71	1.24	1.74	0.78	1.05	1.41
	0.5	1	2	2	0	1	2	1	1	2	0	1	2	0	1	2	0.34	0.54	0.97	0.03	0.39	0.89
	0.7	0	3	16	0	2	16	0	2	16	0	2	16	0	2	16	0.00	1.67	16.39	0.00	1.65	16.39
	0.9	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	0.00	0.00
0.3	0.1	2	3	3	2	3	4	2	3	4	1	2	4	1	2	3	1.02	2.20	3.63	1.02	1.87	2.84
	0.3	3	4	5	2	3	4	2	3	4	2	3	4	2	3	4	1.55	2.22	3.05	1.43	1.81	2.68
	0.5	1	3	5	1	2	4	1	2	4	1	2	4	1	2	3	1.55	1.69	3.88	0.37	1.41	3.64
	0.7	0	1	3	0	1	2	0	1	2	0	0	1	0	1	2	0.00	0.48	1.18	0.00	0.12	0.45
	0.9	0	1	3	0	1	2	0	1	2	0	0	1	0	0	1	0.00	0.32	1.83	0.00	0.16	0.98
0.5	0.1	3	4	5	2	3	4	3	4	5	1	2	3	1	3	4	1.33	2.61	3.54	1.19	2.20	2.90
	0.3	3	6	8	3	4	6	3	5	7	2	3	6	2	4	6	2.58	3.86	5.80	2.11	3.29	5.42
	0.5	2	4	8	2	4	7	2	4	8	1	3	6	1	3	7	1.03	3.58	6.82	1.03	2.90	6.07
	0.7	2	3	5	1	2	4	1	3	6	0	2	4	0	2	4	0.38	2.10	4.05	0.23	1.76	3.82
	0.9	2	4	6	1	2	4	0	2	5	0	1	4	0	1	4	0.00	1.76	4.20	-4.42	1.15	3.96
0.7	0.1	4	7	9	3	5	6	4	5	7	2	4	5	2	4	5	2.42	4.33	5.59	2.42	4.03	4.90
	0.3	4	7	11	4	6	9	4	7	10	3	5	8	3	5	9	3.70	5.48	8.58	2.79	4.67	8.06
	0.5	7	9	13	5	7	11	6	8	12	4	6	11	5	6	11	4.98	6.98	11.92	4.78	6.36	10.76
	0.7	2	9	18	2	8	15	3	8	15	2	7	15	2	7	14	2.92	7.73	15.60	1.39	6.88	14.82
	0.9	6	11	18	4	8	14	4	9	14	3	7	13	3	7	13	3.30	7.94	14.29	3.00	7.52	13.17
0.9	0.1	8	13	18	7	11	17	8	12	19	6	11	16	7	11	16	6.93	11.18	16.77	6.59	10.95	16.84
	0.3	13	17	23	9	14	18	9	14	18	9	13	17	9	13	17	9.28	13.77	18.60	9.28	13.45	17.65
	0.5	10	17	23	3	13	17	4	13	19	3	12	17	3	12	16	3.28	12.56	16.88	3.11	12.27	16.52
	0.7	11	16	21	6	12	18	7	13	18	5	11	17	6	11	17	5.81	11.76	17.98	5.70	11.39	17.72
	0.9	11	16	19	10	12	16	10	13	17	9	12	16	10	12	16	9.27	12.55	15.92	9.86	12.32	16.06
	Avg.	4	7	10	3	5	8	3	5	9	2	5	8	2	5	8	2.54	4.80	8.06	2.14	4.42	7.58

Table 2: Performance of HGA compared to baseline algorithms for $n = 50$

		TS			ABC			DRGA			GA			LOS			HSSGA			HGA		
τ	R	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
0.1	0.1	1	2	3	1	2	2	1	1	2	1	2	3	2	3	5	1.09	1.65	2.48	0.92	1.11	1.34
	0.3	2	2	3	1	1	2	1	1	2	2	2	3	1	3	4	0.81	1.14	1.51	0.71	0.94	1.31
	0.5	0	1	3	0	1	1	1	1	1	1	1	2	1	1	2	0.00	0.37	0.64	0.17	0.31	0.57
	0.7	0	0	1	0	0	0	0	1	1	0	0	0	0	0	2	0.00	0.03	0.30	0.00	0.00	0.00
	0.9	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	0.00	0.00	0.00	0.00
0.3	0.1	1	3	4	1	2	3	2	3	4	2	3	4	4	6	7	1.49	2.18	2.89	0.90	1.53	2.10
	0.3	2	3	5	1	2	4	1	3	4	1	2	4	2	5	7	1.15	1.91	3.36	0.99	1.57	2.86
	0.5	1	2	4	1	2	2	2	2	3	1	2	3	3	4	6	1.15	1.37	2.05	0.68	1.09	1.49
	0.7	0	2	3	0	1	1	1	1	1	0	1	2	1	2	4	0.00	0.52	1.16	0.00	0.25	0.70
	0.9	0	1	2	0	0	1	0	1	1	0	0	1	0	1	3	0.00	0.31	0.74	0.00	0.02	0.19
0.5	0.1	2	4	5	3	3	4	3	5	7	3	4	6	7	8	11	1.96	3.18	4.24	1.96	2.57	3.25
	0.3	3	4	6	2	3	4	4	4	6	3	4	5	6	7	9	2.02	2.98	3.74	1.62	2.37	3.09
	0.5	3	4	5	2	3	4	3	4	6	2	4	5	5	7	10	1.66	3.00	5.11	1.13	2.37	3.32
	0.7	2	3	4	1	2	3	1	3	4	0	2	3	2	5	7	0.77	1.81	2.80	0.31	1.16	1.98
	0.9	1	2	5	1	1	3	1	2	4	0	1	3	2	3	6	0.35	1.35	2.69	0.04	0.64	2.31
0.7	0.1	3	5	6	3	4	5	4	6	8	4	5	6	7	9	11	2.83	4.05	5.23	2.52	3.23	3.99
	0.3	4	7	11	2	5	7	4	6	9	3	5	8	6	9	13	2.19	5.00	7.71	1.71	4.01	6.45
	0.5	4	6	13	3	5	12	4	7	15	3	6	12	6	10	18	2.25	5.54	11.61	2.47	4.62	11.20
	0.7	3	7	13	3	5	7	5	8	12	4	6	9	6	9	13	3.11	6.41	9.68	2.26	5.26	7.77
	0.9	5	8	13	4	6	8	4	7	10	4	6	9	5	9	12	3.36	6.46	9.07	3.10	5.46	7.54
0.9	0.1	7	9	12	5	7	9	6	9	12	5	7	9	8	11	14	5.50	7.15	10.79	4.10	6.25	8.86
	0.3	9	15	23	5	9	12	8	12	16	6	10	13	11	14	18	6.68	10.01	14.39	5.65	9.41	12.04
	0.5	13	16	18	9	12	17	11	14	19	8	12	17	12	16	20	9.84	12.53	18.05	8.91	11.73	16.66
	0.7	11	16	20	8	11	13	10	14	16	9	12	15	11	16	19	8.75	12.18	15.12	8.12	11.57	13.64
	0.9	12	16	22	4	11	17	8	13	19	6	12	17	8	15	22	4.07	11.61	17.79	4.02	11.49	17.27
	Avg.	4	6	8	2	4	6	3	5	7	3	4	6	5	7	10	2.44	4.11	6.13	2.09	3.56	5.20

Table 3: Performance of HGA compared to baseline algorithms for $n = 100$

and expanding our methodology, we anticipate that it will be applicable to a diverse array of real-world problems, particularly those that are characterized by limited resources and complex decision-making processes.

This area of research holds promising possibilities for future exploration. First, dynamic resource allocation mechanisms might be considered to adapt scheduling decisions in real-time in order to take account of varying resource availability. Second, incorporating dynamic order arrivals, where new orders enter the system in real-time, presents an interesting challenge. A dynamic algorithm could be developed that would be able to adapt to these arrivals and optimize schedules on the fly as they occur. Moreover, multi-objective optimization scenarios, in which conflicting objectives such as optimizing revenue, minimizing costs, and ensuring timely delivery coexist, provide an exciting opportunity for further research.

Conflict of interest

The authors declare that they have no conflict of interest.

Data Availability

The data used in this research is a publicly available benchmark dataset that has been used in multiple studies. It can be accessed at https://doi.org/10.4121/uuid:c3623076-a1ac-4103-ad31-3068a28312f9.

References

B. Cesaret, C. Oğuz, and F. S. Salman. A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research*, 39(6):1197–1205, 2012.

S. N. Chaurasia and A. Singh. Hybrid evolutionary approaches for the single machine order acceptance and scheduling problem. *Applied Soft Computing*, 52:725–747, 2017.

L. Davis et al. Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, pages 162–164. Citeseer, 1985.

M. de Weerdt, R. Baart, and L. He. Single-machine scheduling with release times, deadlines, setup times, and rejection. *European Journal of Operational Research*, 291(2):629–639, 2021.

R. Esmaeilbeigi, P. Charkhgard, and H. Charkhgard. Order acceptance and scheduling problems in two-machine flow shops: New mixed integer programming formulations. *European Journal of Operational Research*, 251(2):419–431, 2016.

L. He, A. Guijt, M. de Weerdt, L. Xing, and N. Yorke-Smith. Order acceptance and scheduling with sequence-dependent setup times: A new memetic algorithm and benchmark of the state of the art. *Computers & Industrial Engineering*, 138:106102, 2019.

S.-W. Lin and K.-C. Ying. Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm. *Journal of the Operational Research Society*, 64(2):293–311, 2013.

S. Mahmoudinazlou and C. Kwon. A hybrid genetic algorithm for the min–max multiple traveling salesman problem. *Computers & Operations Research*, 162:106455, 2024.

S. Mahmoudinazlou, A. Alizadeh, J. Noble, and S. Eslamdoust. An improved hybrid ica-sa meta-heuristic for order acceptance and scheduling with time windows and sequence-dependent setup times. *Neural Computing and Applications*, pages 1–19, 2023.

S. Nguyen. A learning and optimizing system for order acceptance and scheduling. *The International Journal of Advanced Manufacturing Technology*, 86:2021–2036, 2016.

S. Nguyen, M. Zhang, and K. C. Tan. A dispatching rule based genetic algorithm for order acceptance and scheduling. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pages 433–440, 2015.

F. T. Nobibon and R. Leus. Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research*, 38(1): 367–378, 2011.

C. Og, F. S. Salman, Z. B. Yalçın, et al. Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, 125(1):200–211, 2010.

J. Park, S. Nguyen, M. Zhang, and M. Johnston. Genetic programming for order acceptance and scheduling. In *2013 IEEE congress on evolutionary computation*, pages 1005–1012. IEEE, 2013.

Y. L. T. Silva, A. Subramanian, and A. A. Pessoa. Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers & operations research*, 90:142–160, 2018.

İ. Tarhan and C. Oğuz. A matheuristic for the generalized order acceptance and scheduling problem. *European Journal of Operational Research*, 299(1):87–103, 2022.

X. Wang, X. Xie, and T. Cheng. A modified artificial bee colony algorithm for order acceptance in two-machine flow shops. *International Journal of Production Economics*, 141(1):14–23, 2013.

X. Wang, G. Huang, X. Hu, and T. Edwin Cheng. Order acceptance and scheduling on two identical parallel machines. *Journal of the Operational Research Society*, 66:1755–1767, 2015.