Chapter 5

Multi-objective Scheduling Optimization

In previous chapters different MOEAs have been proposed for multi-objective optimization. However, the core issue of these algorithms is to solve the real-world application problems, as is required to answer RQ5. The exact real-world problem to be solved is the multi-objective vehicle fleet maintenance scheduling optimization (MOVFMSO) problem. As the preparatory work for solving this real-world scheduling optimization problem, in this chapter, first, study has been done on the benchmark multi-objective scheduling optimization problems, i.e., the flexible job shop scheduling problem (FJSP), and an MOEA has been proposed to solve the FJSP. Therefore, RQ4 is answered. Following this, the real-world MOVFMSO problem is formulated, its representation is defined and the problem specific genetic operators are developed. Based on these works, the previously proposed MOEAs can be applied to solve the MOVFMSO problem.

This chapter continues with Section 5.1, where the FJSP is introduced, the proposed algorithm for solving the FJSP is described and tested. Thereafter, in Section 5.2, the MOVFMSO application problem is established. To solve it, the problem representation, i.e., an encoding of the problem in decision variables, is designed for evolutionary computation. MOEAs have been devised, along with genetic operators. In this chapter, only the basic MOEAs are applied on the MOVFMSO problems, the preference-based MOEA and dynamic MOEA which is developed based on these static algorithms for dynamic environments on the MOVFMSO problem will be introduced in next chapter.

5.1 Tailored NSGA-III Instantiation for Flexible Job Shop Scheduling

In this section, a customized multi-objective evolutionary algorithm is proposed for the multi-objective flexible job shop scheduling problem (MOFJSP) with three objectives (makespan, total workload, critical workload). Multiple initialization approaches have been adopted to produce the first-generation population based on the definition of the chromosome representation; at the same time, diverse genetic operators are applied to guide the search towards offspring with a wide diversity; especially, an algorithm configurator, i.e., the MIP-EGO configurator [110], is used to tune the parameter configuration; furthermore, two levels of local search are employed to explore the neighborhood for better solutions. In general, the proposed algorithm for the FJSP can be combined with any standard MOEAs to solve the MOFJSP. In this work, it has been combined with NSGA-III to solve some benchmark multi-objective FJSPs, whereas an off-the-shelf implementation of NSGA-III is not capable of solving them.

The remainder of this section is structured as follows. The multi-objective FJSP is first introduced in Section 5.1.1. Section 5.1.2 provides necessary background knowledge. In Section 5.1.3, the algorithm strategies for the FJSP are developed, and combined with NSGA-III, therefore, the tailored NSGA-III can solve the multi-objective FJSPs. After that, Section 5.1.4 reports the experimental results. Finally, Section 5.1.5 concludes the work and suggests future work directions.

5.1.1 Flexible Job Shop Scheduling

The job shop scheduling problem (JSP) is an important branch of production planning problems. The classical JSP consists of a set of independent jobs to be processed on multiple machines and each job contains a number of operations with a predetermined order. It is assumed that each operation must be processed on a specific machine with a specified processing time. The JSP is to determine a schedule of jobs, meaning to sequence operations on the machines. The FJSP is an important extension of the classical JSP due to the wide employment of multi-purpose machines in the real-world job shop. The FJSP extends the JSP by assuming that each operation is allowed to be processed on a machine out of a set of alternatives, rather than one specified machine. Therefore, the FJSP is not only to find the best sequence of operations on a machine, but also to assign each operation to a machine out of a set of qualified machines. The JSP is well known to be strongly NP-hard [48]. The FJSP is an even more complex version of the JSP, so the FJSP is clearly also strongly NP-hard.

The MOFJSP addressed in this work is described as follows:

1. There are n jobs $J = \{J_1, J_2, \cdots, J_n\}$ and m machines $M = \{M_1, M_2, \cdots, M_m\}$[1].
2. Each job J_i comprises l_i operations for $i = 1, \cdots, n$, the jth operation of job J_i is represented by O_{ij}, and the operation sequence of job J_i is from O_{i1} to O_{il_i}.
3. For each operation O_{ij}, there is a set of machines capable of performing it, which is represented by M_{ij} and it is a subset of M.
4. The processing time of the operation O_{ij} on machine M_k is predefined and denoted by t_{ijk}.

At the same time, the following assumptions are made:

1. All machines are available at time 0 and assumed to be continuously available.
2. All jobs are released at time 0 and independent from each other.
3. Setting up times of machines and transportation times between operations are negligible.
4. Environmental changes (such as machine breakdowns) are neglected.
5. A machine can only work on one operation at a time.
6. There are no precedence constraints among the operations of different jobs, and the order of operations for each job cannot be modified.
7. An operation, once started, must run to completion.
8. No operation for a job can be started until the previous operation for that job is completed.

The makespan, total workload and critical workload, which are commonly considered in the literature on FJSP (e.g., [16], [131]), are minimized and used as the three objectives in our algorithm. Minimizing the makespan can facilitate the rapid response

[1]In this chapter and the next chapter, m is used to represent the number of machines or workshops to be consistent with prior literature. The objectives have been given for each specific problem.

to the market demand. The total workload represents the total working time of all machines and the critical workload is the maximum workload among all machines. Minimizing the total workload can reduce the use of machines; minimizing the critical workload can balance the workload between machines. Let C_i denote the completion time of job J_i, W_k the sum of processing time of all operations that are processed on machine M_k. The three objectives can be defined as follows:

$$\text{Makespan}(C_{max}) : f_1 = \max\{C_i | i = 1, 2, \cdots, n\} \tag{5.1}$$

$$\text{Total workload}(W_t) : f_2 = \sum_{k=1}^{m} W_k \tag{5.2}$$

$$\text{Critical workload}(W_{max}) : f_3 = \max\{W_k | k = 1, 2, \cdots, m\}. \tag{5.3}$$

An example of MOFJSP is shown in Table 5.1 as an illustration, where rows correspond to operations and columns correspond to machines. In this example, there are three machines: M_1, M_2 and M_3. Each entry of the table denotes the processing time of that operation on the corresponding machine, and the tag " – " means that a machine cannot execute the corresponding operation.

Table 5.1: Processing time of a FJSP instance.

Job	Operation	M_1	M_2	M_3
J_1	O_{11}	3	-	2
	O_{12}	5	7	6
	O_{13}	-	-	2
J_2	O_{21}	2	4	3
	O_{22}	2	-	1
J_3	O_{31}	4	2	2
	O_{32}	3	5	-

5.1.2 Background and Related Work

Algorithms for MOFJSP

The FJSP has been investigated extensively in the last three decades. According to [15], EA is the most popular non-hybrid technique to solve the FJSP. Among all EAs for FJSP, some are developed for the more challenging FJSP: the MOFJSP which we formulated in Section 5.1.1, and [16], [113], [131] are very successful MOFJSP algorithms and have obtained high-quality solutions. [113] proposed a multi-objective genetic algorithm (MOGA) based on the immune and entropy principle. In this MOGA, the fitness was determined by the Pareto dominance relation and the diversity was kept by the immune and entropy principle. In [16], a simple EA (SEA) was proposed, which used domain heuristics to generate the initial population and balanced the exploration and exploitation by refining duplicate individuals with mutation operators. A memetic algorithm (MA) was proposed in [131] and it incorporated a local search into NSGA-II [29]. A hierarchical strategy was adopted in the local search to handle objectives. In Section 5.1.4, these algorithms have been compared with the proposed algorithm on the benchmark MOFJSPs.

Parameter Tuning

EA involves using multiple parameters, such as the crossover probability, mutation probability, computational budget, as so on. The preset values of these parameters affect the performance of the algorithm in different situations. The parameters are usually set to values which are assumed to be good. For example, the mutation probability normally is kept very low, otherwise the convergence is supposed to be delayed unnecessarily. But the best way to identify the probability would be to do a sensitivity analysis: carrying out multiple runs of the algorithms with different mutation probabilities and comparing the outcomes. Although there are some self-tuning techniques for adjusting these parameters on the go, the hyper-parameters in EA can be optimized using the technique from machine learning.

The optimization of hyper-parameters and neural network architectures is a very important topic in the field of machine learning due to the large number of design choices for a network architecture and its parameters. Recently, algorithms have been developed to accomplish this automatically since it is intractable to do it by hand. The MIP-EGO [110] is one of these configurators that can automatically configure convolutional neural network architectures and the resulting optimized neural networks have been proven to be competitive with the state-of-the-art manually designed ones on some popular classification tasks. Moreover, MIP-EGO allows for multiple candidate points to be selected and evaluated in parallel, which can speed up the automatic tuning procedure. In this work, several parameters are tuned with MIP-EGO to find the best parameter setting for them.

NSGA-III

NSGA-III is a decomposition-based MOEA, it is an extension of the well-known NSGA-II and eliminates the drawbacks of NSGA-II such as the lack of uniform diversity among a set of non-dominated solutions. The basic framework of NSGA-III is similar to the original NSGA-II, while it replaces the crowding distance operator with a clustering operator based on a set of reference points. A widely-distributed set of reference points can efficiently promote the population diversity during the search and NSGA-III defines a set of reference points by Das and Dennis′s method [21].

In each iteration t, an offspring population Q_t of size N_{pop} is created from the parent population P_t of size N_{pop} using usual selection, crossover and mutation. Then a combined population $R_t = P_t \cup Q_t$ is formed and classified into different layers (F_1, F_2, and so on), each layer consists of mutually non-dominated solutions. Thereafter, starting from the first layer, points are put into a new population S_t. A whole population is obtained until the first time the size of S_t is equal to or larger than N_{pop}. Suppose the last layer included in S_t is the l-th layer, so far, members in $S_t \setminus F_l$ are points that have been chosen for P_{t+1} and the next step is to choose the remaining points from F_l to make a complete P_{t+1}. In general (when the size of S_t doesn't equal to N_{pop}), $N_{pop} - |S_t \setminus F_l|$ solutions from F_l needs to be selected for P_{t+1}.

When selecting individuals from F_l, first, each member in S_t is associated with a reference point by searching the shortest perpendicular distance from the member to all reference lines created by joining the ideal point with reference points. Next, a niching strategy is employed to choose points associated with the least reference points in P_{t+1} from F_l. The niche count for each reference point, defined as the number of members in $S_t \setminus F_l$ that are associated with the reference point, is computed. The member in F_l associated with the reference point having the minimum niche count is included in P_{t+1}. The niche count of that reference point is then increased by one and the procedure is repeated to fill the remaining population slots of P_{t+1}.

NSGA-III is powerful to handle problems with non-linear characteristics as well as having many objectives. Therefore, we decided to enhance NSGA-III in our algorithm for the MOFJSP.

5.1.3 Proposed Algorithm

The proposed algorithm, *Flexible Job shop Scheduling Problem Multi-Objective Evolutionary Algorithm* (FJSP-MOEA) can in principal be combined with any MOEA and help MOEAs solve the MOFJSP, whereas the standard MOEAs cannot solve MOFJSP solely. The algorithm follows the flow of a typical EA and generates improved solutions by using local search. Details of the following components are given in the next subsections.

- Initialization: encode the individual and generate the initial population.
- Genetic operators: generate offspring by crossover and mutation operators.
- Local search: decode the individual and improve the solution with local search.

Initialization

1. Chromosome Encoding

The MOFJSP is a combination of assigning each operation to a machine and ordering operations on the machines. In the algorithm, each chromosome (individual) represents a solution in the search space and the chromosome consists of two parts: the operation sequence vector and the machine assignment vector. Let N denote the number of all operations of all jobs. The length of both vectors is equal to N. The operation sequence vector decides the sequence of operations assigned to each machine. For any two operations which are processed by the same machine, the one located in front is processed earlier than the other one. The machine assignment vector assigns the operations to machines, in other words, it determines which operation is processed by which machine and the machine should be the one capable of processing the operation.

The format of representing an individual not only influences the implementation of crossover and mutation operators, a proper representation can also avoid the production of infeasible schedules and reduces the computational time. In the algorithm, the chromosomal representation proposed by Zhang et al. in [134] is adopted and an example is given in Table 5.2.

Table 5.2: An example of a chromosome representation.

Operation sequence	**1**	**2**	**3**	**2**	**1**	**1**	**3**
	O_{11}	O_{21}	O_{31}	O_{22}	O_{12}	O_{13}	O_{32}
Machine assignment	**2**	**1**	**1**	**3**	**2**	**2**	**1**
	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{31}	O_{32}
	M_3	M_1	M_3	M_3	M_3	M_2	M_1

In Table 5.2, the first row shows the operation sequence vector which consists of only job indexes. For each job, the first appearance of its index represents the first operation of that job and the second appearance of the same index represents the second operation of that job, and so on. The occurrence number of an index is equal to the number of operations of the corresponding job. The second row explains the first row by giving the real operations. The third row is the machine assignment vector which presents the selected machines for all operations. The operation sequence of the machine assignment vector is fixed, which is from the first job to the last job and from the first operation to the last operation for each job. The fourth row indicates the fixed operation sequence of the machine assignment vector and the fifth row shows the real machines of the operations. Each integer value in the machine assignment vector is the index of the machine in the set of alternative machines of that operation. In this example, O_{13} is assigned to M_3 because M_3 is the first (and only) machine in the alternative machine set of O_{13} (Table 5.1). The alternative machine set of O_{22} is $\{M_1, M_3\}$, the second machine in this set is M_3, therefore, O_{22} is assigned to M_3.

2. Initial Population

The algorithm starts by creating the initial population. The machine assignment and operation sequence vectors are generated separately for each individual. In the literature, a few approaches have been proposed for producing individuals, such as global minimal workload in [67]; AssignmentRule1 and AssignmentRule2 in [82]. In the proposed algorithm, several new methods are proposed, namely the *Processing Time Roulette Wheel* (PRW) and *Workload Roulette Wheel* (WRW) for initialising the machine assignment and the *Most Remaining Machine Operations* (MRMO) and *Most Remaining Machine Workload* (MRMW) for initialising the operation sequence. These new approaches have been used together with some commonly used dispatching rules in initializing individuals for the purpose of enriching the initial population. When generating a new individual, two initialization methods are randomly picked from the following two lists; one for the machine assignment vector and one for the operation sequence vector.

Initialization Methods for Machine Assignment

1. Random assignment (Random): an operation is assigned to an eligible machine randomly.
2. Processing time Roulette Wheel (PRW): for each operation, the roulette wheel selection is adopted to select a machine from its machine set based on the processing times of these capable machines. The machine with the shorter processing time is more likely to be selected.
3. Workload Roulette Wheel (WRW): for each operation, the roulette wheel selection is used to select a machine from its machine set based on the current workloads

plus the processing times of these capable machines. The machine with lower sum of the workload and processing time is more likely to be selected.

PRW and WRW are proposed to assign the operation to the machine with less processing time or accumulated workload, at the same time, maintaining the freedom of exploring the entire search space.

Initialization Methods for Operation Sequence

1. Random permutation (Random): starting from a fixed sequence: all job indexes of J_1 (the number of J_1 job indexes is the number of operations of J_1), followed by all job indexes of J_2, and so on. Then the array with the fixed sequence is permuted and a random order is generated.
2. Most Work Remaining (MWR): operations are placed one by one into the operation sequence vector. Before selecting an operation, the remaining processing times of all jobs are calculated respectively, the first optional operation of the job with the longest remaining processing time is placed into the chromosome.
3. Most number of Operations Remaining (MOR): operations are placed one by one into the operation sequence vector. Before selecting an operation, the number of succeeding operations of all jobs is counted respectively, the first optional operation of the job with the most remaining operations is placed into the chromosome.
4. Long Processing Time (LPT)[127]: operations are placed one by one into the operation sequence vector, each time, the operation with maximal processing time is selected without breaking the order of jobs.
5. Most Remaining Machine Operations (MRMO): operations are placed into the operation sequence vector according to both the number of subsequent operations on machines and the number of subsequent operations of jobs. MRMO is a hierarchical method and takes the machine assignment into consideration. First, the machine with the most subsequent operations is selected. After that, the optional operations in the subsequent operations on that machine are found based on the already placed operations. For example, if $O_{11} \rightarrow O_{12} \rightarrow O_{21}$ are placed operations, the current optional operation can only be chosen from O_{13}, O_{22}, and O_{31}. In these optional operations, those which are assigned to the selected machine are picked and the one that belongs to the job with the most subsequent operations is placed into the chromosome. In this example, O_{31} will be chosen if it is assigned to the selected machine because there are two subsequent operations for J_3 and only

one subsequent operation for J_1 and J_2. Note that it is possible that no operation is available on that machine, in that case, the machine with the second biggest number of subsequent operations will be selected, and so forth.

6. Most Remaining Machine Workload (MRMW): operations are placed into the operation sequence vector according to both the remaining processing times of machines and the remaining processing times of jobs. MRMW is a hierarchical method similar to MRMO. After finding the machine with the longest remaining process time and the optional operations on that machine, the operation which belongs to the job with the longest remaining process time is placed into the chromosome. Again, if no operation is available on that machine, the machine with the second longest remaining processing time will be selected, and so forth.

MRMO and MRMW are proposed to give priority to both the machine and the job with the most number of remaining operations (MRMO) and the longest remaining processing time (MRMW).

Crossover

Crossover is a matter of replacing some of the genes in one parent with the corresponding genes of the other. Since the representation of chromosomes has two parts, crossover operators applied to these two parts of chromosomes are implemented separately as well. Two new crossover operators, *Precedence Preserving Two Points Crossover* (PPTP) and *Uniform Preservative crossover* (UPX), are proposed and used together with several commonly adopted crossover operators. When executing the crossover operation in the proposed algorithm, one crossover operator for machine assignment and one operator for the operation sequence, are randomly chosen from the following two lists to generate the offspring.

Crossover Operators for Machine Assignment

1. No crossover
2. One point crossover: a cutting point is picked randomly and genes after the cutting point are swapped between two parents.
3. Two points crossover: two cutting points are picked randomly and genes between the two points are swapped between two parents.
4. Job-based crossover (JX): it generates two children from two parents by the following procedure:

  a A vector with the size of the jobs is generated, which consists of random values 0 and 1.
  b For the job corresponding to value 0, the assigned machines of its operations are preserved.
  c For the job corresponding to value 1, the machines of its operations are swapped between two parents.

5. Multi-point preservative crossover (MPX)[133]: MPX generates two children from two parents by the following procedure:

  a A vector with the size of all operations is generated, which consists of random values 0 and 1.
  b For the operations corresponding to value 0, their machines (genes) are preserved.
  c For the operations corresponding to value 1, their machines (genes) are swapped between the two parents.

Crossover Operators for Operation Sequence

1. No crossover
2. Precedence preserving one point crossover (PPOP) [102]: PPOP generates two children from two parents by the following procedure:

  a A cutting point is picked randomly, genes to the left are preserved and copied from parent1 to child1 and from parent2 to child2.
  b The remaining operations in parent1 are reallocated in the order they appear in parent2.
  c The remaining operations in parent2 are reallocated in the order they appear in parent1.

   An example of PPOP is shown in Figure 5.1 and the cutting point is between the third and fourth operation. Red numbers in parent2 are the genes on the right side of the cutting point in parent1 and they are copied to child1 with their own sequence following the genes on the left side of the cutting point in parent1, and vice versa.
3. Precedence Preserving Two Points Crossover (PPTP): PPTP generates two children from two parents by the following procedure:

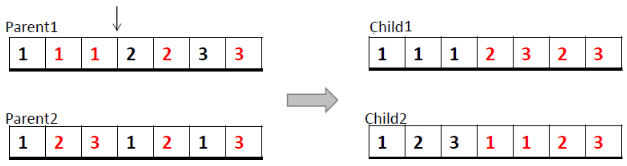

Figure 5.1: The process of PPOP.

a Two cutting points are picked randomly, genes except for those between the two points are preserved and copied from parent1 to child1 and from parent2 to child2.

b Operations between the two cutting points in parent1 are reallocated in the order they appear in parent2.

c Operations between the two cutting points in parent2 are reallocated in the order they appear in parent1.

4. Improved precedence operation crossover (IPOX)[132]: IPOX divides the job set into two complementary and non-empty subsets randomly. The operations of one job subset are preserved, while the operations of another job subset are copied from another parent.

5. Uniform Preservative crossover (UPX): UPX generates two children from two parents by the following procedure:

   a A vector with the size of all operations is generated, which consists of random values 0 and 1.

   b For the operations corresponding to value 0, the genes are preserved and copied from parent1 to child1 and from parent2 to child2.

   c For the operations corresponding to value 1, the genes in parent1 are found in parent2 and copied from parent2 with the sequence in parent2, and vice versa.

Mutation

The mutation operator flips the gene values at selected locations. By forcing the algorithm to search areas other than the current area, the mutation operator is used to maintain genetic diversity from one generation of a population to the next. In this algorithm, insertion mutation and swap mutation (including one point swap and two points swap) are proposed and used.

Insertion Mutation Operator generates a new individual by the following procedure:

- Two random numbers i and j ($1 \leq i \leq N$, $1 \leq j \leq N$) are selected.
- For the operation sequence vector, the operation on position j is inserted in front of the operation on position i.
- For the machine assignment vector, a machine is randomly selected for both the operations on i and on j respectively. If the processing time on the newly selected machine is lower than that on the current machine, the current machine is replaced by the new machine. If the processing time on the new machine is longer than that on the old machine, there is only a 20% probability that the new machine replaces the old machine.

Swap Mutation Operator generates a new individual by the following procedure:

- One random number i ($1 \leq i \leq N$) is selected or two random numbers i and j ($1 \leq i \leq N$, $1 \leq j \leq N$) are selected.
- For the operation sequence vector, with only one swap point i, the operation on the swap point is swapped with its neighbour; with two swap points, the operations on position i and j are swapped.
- For the machine assignment vector, the machine on position i (and j) is replaced with a new machine by the same rule used in the insertion mutation operator.

Decoding and Local Search

Decoding a chromosome is to convert an individual into a feasible schedule to calculate the objective values which represents the relative superiority of a solution. In this process, the operations are picked one by one from the operation sequence vector and placed on the machines from the machine assignment vector to form the schedule. When placing each operation to its machine, local search (in the sense of heuristic rules to improve solution) is involved to refine an individual in order to obtain an improved schedule in the proposed algorithm. Two levels of local search are applied to allocate each operation to a time slot on its machine. We know that idle times may exist between operations on each machine due to precedence constraints among operations of each job, and two levels of local search utilize idle times in different degrees.

The First Level Local Search

Let S_{ij} be the starting time of O_{ij} and C_{ij} the completion time of O_{ij}, an example of the first level local search is shown in Figure 5.2. Because O_{mn} needs to be processed after the completion of O_{mn-1}, an idle time interval between the completion of O_{ab} and the starting of O_{mn} appeared on machine M_k. O_{ij} is assigned to M_k and we assume that O_{mn} is the last operation on M_k before handling O_{ij}, therefore the starting time of O_{ij} is $\max\{C_{mn}, C_{ij-1}\}$, which in this example is C_{mn} and it is later than C_{ij-1}, thus, there is an opportunity that O_{ij} can be processed earlier. When checking the idle time on M_k, the idle time interval $[C_{ab}, S_{mn}]$ is found available for O_{ij} because the idle time span $[C_{ij-1}, S_{mn}]$, which is part of $[C_{ab}, S_{mn}]$, is enough to process O_{ij} or longer than t_{ijk}.

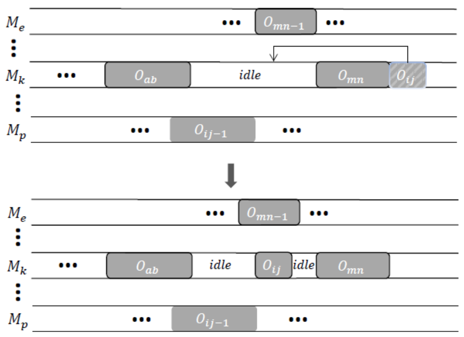

Figure 5.2: First level local search

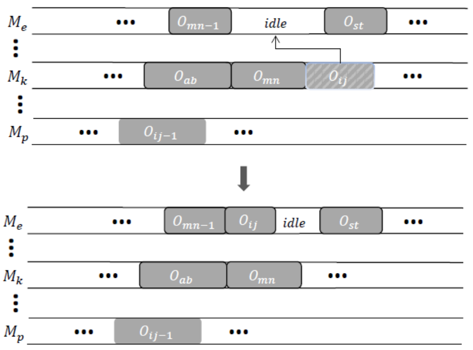

Figure 5.3: Second level local search

Let S_k^d be the starting time of the dth idle time interval on M_k and C_k^d be the completion time. O_{ij} can be transferred to an earliest possible idle time interval of its machine which satisfies the following equation:

$$\max\{S_k^d, C_{ij-1}\} + t_{ijk} \leq C_k^d, (C_{ij} = 0, \text{if } j = 1). \tag{5.4}$$

After using the idle time interval, the starting time of O_{ij} is $\max\{S_k^d, C_{ij-1}\}$ and the idle interval is updated based on the starting and completion time of O_{ij}: (1) the idle time interval is removed; (2) the starting or completion time of the idle time interval is modified; (3) the idle time interval is replaced by two new shorter idle time intervals, like in the example of Figure 5.2.

After decoding a chromosome, the operation sequence vector of the chromosome is updated according to new starting times of operations, and three objective values are calculated. The first level local search only finds for each operation the available idle time interval on its assigned machine. After generating the corresponding schedule with the first level search method, it is possible that there are still operations that can be allocated to available idle time intervals to benefit the fitness value. To achieve this, decoding the chromosome which has been updated with the first level local search is performed with the second level local search, and again operations are moved to available idle time intervals.

The Second Level Local Search

The second level local search not only checks the idle time intervals on the assigned machine, but also the idle time intervals on alternative machines. An example of making use of the idle time interval on another machine is shown in Figure 5.3. Let S_{ijk} be the starting time and C_{ijk} be the completion time of O_{ij} on M_k. In this example, O_{ij} is assigned to M_k in the initial chromosome, we assume that O_{ij} can also be performed by M_e. Under the condition that the starting time of O_{ij} on M_k is later than the completion time of O_{ij-1}, the idle time intervals on all alternative machines which can process O_{ij} are checked. An idle time interval on M_e could be a choice and O_{ij} can be reallocated to M_e. In this example, the processing time of O_{ij} on M_e is even shorter then the processing time on M_k, therefore, this reallocation can at least benefit the total workload.

In the second level local search, all available idle time intervals of an operation are checked one by one until the first "really" available idle time interval is found and then the operation is moved to that idle time interval. Any idle time interval on an alternative machine which can satisfy Equation 5.4 is an available idle time interval, while it must meet at least one of the following conditions to become a "really" available idle time interval.

1. The processing time of the operation on the new machine is shorter than on the initially assigned machine if the available idle time interval is on a different machine;
2. The operation can be moved from the machine with the maximal makespan to another machine.
3. The operation can be moved from the machine with the maximal workload to another machine.

The total workload can be improved directly by the first condition; the motive of the second condition is to decrease the maximal makespan and the third condition can benefit the critical workload.

After the reallocation of the operations with the second level local search, the corresponding schedule is obtained and objective values are calculated. While, instead of updating the chromosome immediately, the new objective values are compared with the old objective values first, the chromosome can be updated only when at least one objective is better than its old value. This is to make sure that the new schedule is at least not worse than the old schedule (The new solution is not dominated by the old solution). Another difference between the first and second level local search is that the first level local search is performed on every evaluation, while the second level local search is only performed with a 30% probability for each chromosome to avoid local optima. Although these two local searches can be applied repeatedly to improve the solution, to avoid that the algorithm is stuck in a local optima, they are employed only once for each evaluation.

5.1.4 Experimental Results

The algorithms are tested on two sets of well-known FJSP benchmark instances: 4 Kacem instances (ka4x5, ka10x7, ka10x10, ka15x10) and 10 BRdata instances (Mk01-Mk10). Table 5.3 gives the scale of these instances. The first column is the name of each instance; the second column shows the size of the instance, in which n stands for the number of jobs and m the number of machines; the third column represents the number of operations; the fourth column lists the flexibility of each instance, which means the average number of alternative machines for each operation in the problem.

All the experiments are performed with a population size of 100, each run of the algorithm will stop based on a predefined number of evaluation, which is 10000 for Kacem instances and 150000 for BRdata instances. For each problem instance, the proposed algorithm is independently run 30 times. The resulting set is formed by all non-dominated solutions from the union of 30 runs.

The crossover probability is set to 1 and two random crossover operators can be chosen each time (one for operation sequence and one for machine assignment). For Kacem instances, the mutation probabilities are set to 0.6. For BRdata instances, which include larger-scale and more complex problems, the MIP-EGO configurator [110] is adopted to tune both insertion and swap mutation probabilities (one point swap mutation and two points swap mutation) to find the best parameter values for each problem. The hypervolume of the solution set has been used in MIP-EGO as the

Table 5.3: The scale of benchmark instances.

Instance	n → m	#Opr	Flex.
ka4x5	4 → 5	12	5
ka10x7	10 → 7	29	7
ka10x10	10 → 10	30	10
ka15x10	15 → 10	56	10
Mk01	10 → 6	55	2
Mk02	10 → 6	58	3.5
Mk03	15 → 8	150	3
Mk04	15 → 8	90	2
Mk05	15 → 4	106	1.5
Mk06	10 → 15	150	3
Mk07	20 → 5	100	3
Mk08	20 → 10	225	1.5
Mk09	20 → 10	240	3
Mk10	20 → 15	240	3

objective value to tune three mutation probabilities. Although the true PFs for test instances are unknown, [131] provides the reference set for Kacem and BRdata FJSP instances, which is formed by gathering all non-dominated solutions found by all the implemented algorithms in [131] and also non-dominated solutions from other state-of-the-art MOFJSP algorithms. The reference point for calculating the hypervolume value is determined by the largest value in this reference set. To be specific, each objective function value of the reference point is: 1.1 $\times$ largest objective function value of the respective dimension in the reference set. The origin point is used as the ideal point. Other basic parameter settings of MIP-EGO are listed in Table 5.4. For each mutation probability, we only consider a discretized number with only one digit after the decimal point, therefore, the search space is ordinal or integer space, which in MIP-EGO are handled in the same way.

Table 5.4: Settings for MIP-EGO.

Parameter	value
maximal number of evaluations	200
surrogate model	random forest
optimizer for infill criterion	MIES
search space	ordinal space

Table 5.5 shows the percentage of the evaluations which can achieve the largest hypervolume value (or the best PF) by MIP-EGO (200 Evaluations). In each evaluation, MIP-EGO assigns a specific parameter setting for our optimization algorithm. It

can be observed for Mk05 and Mk08 that all the evaluations have obtained the largest hypervolume value; it means that all parameter values of mutation probabilities in MIP-EGO can achieve the best PF for these two problems. It can also be seen in Table 5.3 that both problems have a low flexibility value. On the contrary, for Mk06, Mk09 and Mk10, these problems have a large operation number and high flexibility. It seems that they can be difficult to solve because only one best parameter setting for mutation probabilities has been found among all evaluations. This also means that it is highly likely better solution sets can be found with a higher budget.

Table 5.5: Probability of finding best configuration.

Mk01	Mk02	Mk03	Mk04	Mk05	Mk06	Mk07	Mk08	Mk09	Mk10
73%	60%	95%	1%	100%	0.5%	4.5%	100%	0.5%	0.5%

With the best parameter setting of the mutation probabilities for BRdata instances, our experimental results are compared with the reference set in [131]. The proposed algorithm can achieve the same Pareto optimal solutions as in the reference set for all BRdata instances except for Mk06, Mk09 and Mk10. At the same time, for Mk06 and Mk10, our algorithm can find new non-dominated solutions. Table 5.6 is the list of new non-dominated solutions obtained by our algorithm, each row of an instance is a solution with three objectives: makespan, total workload, and critical workload.

Table 5.6: Newly achieved non-dominated solutions.

Mk06			Mk10		
61	427	53	218	1973	195
63	428	52	218	1991	194
63	435	51	219	1965	195
65	453	49	220	1984	191
66	451	49	225	1979	194
66	457	48	226	1954	196
			226	1974	194
			226	1979	192
			228	1973	194
			235	1938	199
			236	1978	193

Another comparison is between our algorithm (FJSP-MOEA) and MOGA [113], SEA [16] and MA1, MA2 [131]. In [131], there are several variants of the proposed algorithm with different strategies in the local search. MA1 and MA2 are chosen as compared algorithms because they perform equally good or superior to other variants on almost all problems. Table 5.7 displays the hypervolume value of the PF approximation from all algorithms and the new reference set which is formed by combining all solutions from the PF by all algorithms. The highest hypervolume value on each problem in all algorithms has been highlighted in bold. It can be observed that FJSP-MOEA and MA1, MA2 show the best and similar performance, and MOGA behaves the best for three of the BRdata instances. The good performance of MOGA on three problems is interesting. MOGA has an entropy-based mechanism to maintain decision space diversity which might be beneficial for solving these problem instances. When using one best parameter setting, the average hypervolume and standard deviation from 30 runs on each problem are given in Table 5.8, the standard deviation of each problem shows the stable behaviour of each run.

Table 5.7: Hypervolume from MOGA, SEA, MA1, MA2, FJSP-MOEA and the reference set.

Problem	MOGA	SEA	MA1	MA2	FJSP-MOEA	Ref
Mk01	0.00426	0.00508	**0.00512**	**0.00512**	**0.00512**	0.00512
Mk02	0.01261	0.01206	**0.01294**	**0.01294**	**0.01294**	0.01294
Mk03	**0.02460**	0.02165	0.02165	0.02165	0.02165	0.02809
Mk04	**0.06906**	0.06820	0.06901	0.06901	0.06901	0.07274
Mk05	0.00626	0.00635	**0.00655**	**0.00655**	**0.00655**	0.00655
Mk06	0.05841	0.06173	0.06585	0.06692	**0.06709**	0.07065
Mk07	0.02244	0.02132	**0.02269**	**0.02269**	**0.02269**	0.02288
Mk08	**0.00418**	0.00356	0.00361	0.00361	0.00361	0.00428
Mk09	0.01547	0.01755	0.01788	**0.01789**	0.01785	0.01789
Mk10	0.01637	0.01778	0.02145	**0.02196**	0.02081	0.02249

Table 5.8: Average hypervolume and std with the best parameter setting.

Problem	Mk01	Mk02	Mk03	Mk04	Mk05	Mk06	Mk07	Mk08	Mk09	Mk10
Ave-HV	0.0050	0.0122	0.0216	0.0672	0.0064	0.0598	0.0222	0.0036	0.0174	0.0186
Std	0	0.0003	0.0001	0.0004	0.0001	0.0019	0.0003	0	0.0002	0.0006

For Kacem instances and with fixed mutation probabilities, the obtained non-dominated solutions by the proposed algorithm are the same as the PF in the reference set. MA1 and MA2 also achieved the best PF for all Kacem instances, but the proposed FJSP-MOEA uses far less computational resources. It uses only a population size of 100 whereas the population size of MA algorithms is 300. FJSP-MOEA uses only 10000 objective function evaluations, whereas MA uses 150000 evaluations. In terms of computational resources the proposed FJSP-MOEA can therefore be used on smaller computer systems, entailing broader applicability, and possibly also in real-time algorithm implementations such as dynamic optimization.

5.1.5 Conclusion

A novel multi-objective evolutionary algorithm for the multi-objective flexible job shop scheduling problem (MOFJSP) is proposed. It uses multiple initialization approaches to enrich the first generation population, and various crossover operators to create better diversity for offspring. Moreover, to determine the optimal mutation probabilities, the MIP-EGO configurator is adopted to automatically generate proper mutation probabilities. Besides, the straightforward local search is employed with different levels to aid more accurate convergence to the PF. The proposed customization approach in principle can be combined with almost all MOEAs. In this work, it is incorporated with one of the state-of-the-art MOEAs, namely NSGA-III, to solve the MOFJSP, and the new algorithm can find all Pareto optimal solutions in literature for most problems, and even new Pareto optimal solutions for the large scale instances.

The ability of the MIP-EGO configurator in finding the optimal mutation probabilities is shown in this work. However, there is more potential in the automated parameter configuration domain that can benefit EA. For example, to know the effects of different initialization approaches and crossover operators, we can optimize the initialization and crossover configuration. Furthermore, other parameters of the proposed algorithm, such as, population size, evaluation number, and so on, can also be tuned automatically. However, so far the efficiency of the existing tuning framework is limited when it comes to a larger number of parameters. It would therefore be a good topic of future research to find more efficient implementations of these.

5.2 MOEAs for Vehicle Fleet Maintenance Scheduling Optimization

Nowadays, companies, corporations, and organizations of all sorts rely on vehicle fleets to deliver products and services. Typical examples of such vehicle fleets are taxi cab fleets, public bus fleets, car rental fleets, delivery fleets, and so on. According to the statistical data from the European Automobile Manufacturers Association, the global vehicle fleet grows continuously, and the EU has a total fleet of 259.7 million passenger cars and 39.1 million commercial vehicles in 2017[2]. The maintenance of fleet vehicles plays a critical role in their efficient use. Fleet vehicles should be maintained according to a schedule to ensure that they are safe for use; at the same time, a good maintenance schedule can reduce related expenses, increase the efficiency of the assets, ensure consistent service delivery, and even reduce its carbon footprint.

Due to various tasks the vehicles execute, damages to vehicles occur after varying duration. To be more precise, the critical components of each car need to be maintained based on their respective damages from wear and tear. To decide when each component should be maintained, the remaining useful lifetime (RUL) of each component, which is the time remaining until the component no longer meets operational requirements, can be predicted based on adequate predictive approaches or models [35]. Other than the predicted RUL of components, to maintain a vehicle fleet, different maintenance resources are needed. For example, a vehicle component may take several days to be repaired and the maintenance activity can be performed on one of several optional workshops.

The goal of optimizing the maintenance schedule for a vehicle fleet is to ensure that all maintenance tasks are performed on time, keeping the vehicle fleet in healthy operating condition and under business requirements. This section continues in Section 5.2.1 with the formulation of the vehicle fleet maintenance scheduling optimization (VFMSO) problem. A literature review is provided in Section 5.2.2. The customized multi-objective evolutionary algorithm for the VFMSO problem is described in Section 5.2.3, Section 5.2.4 presents and discusses experiments and their results. Lastly, Section 5.2.5 concludes the work and outlines directions for future work.

5.2.1 Problem Formulation

In the real world, the size of a vehicle fleet can be large and its distribution wide, which makes it necessary to distribute the maintenance of a vehicle fleet in multiple separate workshops. At the same time, each workshop has its own capacity and ability, meaning that on the one hand, each workshop has its own team and each team can work on only one car simultaneously; on the other hand, each workshop is limited to the maintenance of the specific component(s) due to restrictions in the equipment or skill level of the staff. These conditions form the primary constraints faced by the vehicle fleet maintenance scheduling optimization. Moreover, the cost and time which are needed to repair car components by different workshops are required. It

[2]https://www.acea.be/statistics/tag/category/size-distribution-of-vehicle-fleet retrieved on 17th of December 2018.

is possible that the maintenance of the same component produces different costs and workloads when the activity is performed in different workshops because, for example, the distances between vehicles and the workshops are different. The set-up cost and set-up time are fixed for each visit of a car to a workshop, which correspond to the cost and time required for the preparation of the maintenance. In this work, the predicted RULs of components are converted to the due dates to determine the maintenance time of each component which is the estimate of the date when the component fails in case no maintenance takes place before its due date.

The vehicle fleet maintenance scheduling optimization problem addressed in this work is defined as follows:

1. There are n cars $C = \{C_1, C_2, \cdots, C_n\}$ and m workshops $W = \{W_1, W_2, \cdots, W_m\}$.
2. Each car C_i comprises l_i components to be maintained for $i = 1, \cdots, n$.
3. For each component O_{ij}, i.e., the jth operation of car C_i, there is a set of workshops capable of repairing it. The set of workshops is represented by W_{ij} which is a subset of W.
4. The processing time of the maintenance of the component O_{ij} in workshop W_k is predefined and denoted by p_{ijk}.
5. The maintenance cost of the maintenance of the component O_{ij} in workshop W_k is predefined and denoted by q_{ijk}.
6. The set-up time of car C_i in workshop W_k is predefined and denoted by X_{ik}.
7. The set-up cost of car C_i in workshop W_k is predefined and denoted by Y_{ik}.
8. The number of teams in workshop W_k is predefined and denoted by Z_k.
9. The due date and previous repair time of each component O_{ij} are predefined and denoted by D_{ij} and R_{ij} respectively.
10. All business requirements or vehicle demands are predefined. There are r vehicle demands and the format of one demand is: N_i cars are required from day d_{i1} to day d_{i2} for $i = 1, \cdots, r$.

At the same time, the following assumptions are made:

1. All workshops and cars are available at time 0 and assumed to be continuously available.
2. All the components are independent from each other.

3. Times required for transport of cars from/to workshops are included in the maintenance time and cost of cars.

4. Environmental changes (such as car accidents) are not considered here.

5. There are no precedence constraints among the components of different cars. Cars are maintained on a first-come-first-served basis.

6. An operation, once started, must run to completion.

7. No operation can start before completion of the previous operation.

A multi-objective scheduling optimization problem is considered in this work and three objectives are assumed to be relevant for the vehicle fleet operator, which are the total workload, total cost and demand satisfaction. The reason why demand satisfaction is defined as objective is that it is treated as flexible and violable. Let T_k denote the sum of the maintenance times spent on all operations that are processed in workshop W_k; M_i the sum of all costs of all operations of car C_i; N^t_{avail} the number of cars which are not in workshops on day t, N^t_{demand} the number of cars required on day t. Three objectives can be defined as:

$$\text{Minimize the total workload : } f_1 = \sum_{k=1}^{m} T_k \tag{5.5}$$

$$\text{Minimize the total cost : } f_2 = \sum_{i=1}^{n} M_i \tag{5.6}$$

$$\text{Maximize the demand satisfaction : } f_3 = \sum_{t} \min\{N^t_{avail} - N^t_{demand}, 0\}. \tag{5.7}$$

For illustration purposes, an example of parameters for a car is shown in Table 5.9, where rows correspond to components of the car; columns correspond to the cost, processing time, set-up time and set-up cost of the car in alternative workshops, also due date and previous repair date. In this example, there are two workshops: W_1 and W_2. The tag " – " means that the workshop cannot repair the corresponding component, therefore, workshop W_1 cannot maintain O_{14} and workshop W_2 cannot maintain O_{11}. Here, the predefined costs of components are the same at both workshops: €200, so are the processing times, set-up costs and set-up times. Of course, times and costs in different workshops can be different. The due date and previous repair time are relative to day 0.

Table 5.9: Parameters of car C_1.

Comp	cost		time		date		set-up cost		set-up time	
	W_1	W_2	W_1	W_2	Due	Pre	W_1	W_2	W_1	W_2
O_{11}	200	-	2	-	74	-61	100	1	100	1
O_{12}	200	200	2	2	25	-6				
O_{13}	200	200	2	2	15	-50				
O_{14}	-	200	-	2	60	-1				

5.2.2 Literature Review

In the early studies of predictive maintenance, also called condition-based maintenance (CBM) [63], the condition monitoring, failure diagnostics, and prognostics always attracted more attention than planning the maintenance schedule based on the information obtained from the condition monitoring, failure diagnostics and prognostics. In [83], an onboard locomotive diagnostic system was invented to monitor the transmitted onboard vehicle data, determine whether any of the monitored data is out of a predetermined range, compare monitored data with historical data and calculate trends, predict if any vehicle system(s) must be corrected to avoid vehicle failure and when such system(s) are likely to fail. It was mentioned in [83] that the onboard diagnostic systems are not helpful in optimizing locomotive maintenance scheduling because they do not communicate with a rail carrier's scheduling center.

Simultaneously, the prognostic information included in maintenance policies may be given in different formats such as RUL, efficiency decrease, the probability of failure. Prognostics and reliability information in [14] were the failure probabilities of the components. The failure probabilities were analyzed to schedule maintenance with minimum system risk using a genetic algorithm. The systematic risk was treated as part of the cost of performing the maintenance schedule. System conflicts and resources were also considered as constraints in this single-objective problem. [10] introduced deterioration models to take into account the component degradation information of a multi-component system. The optimal dates of maintenance operations can be computed based on the model, and additional costs can be evaluated if the operations are not executed at the optimal maintenance dates. Lastly, the optimal grouping of individual maintenance actions was found to reduce the maintenance cost, and the maintenance plan was updated dynamically at each inspection date.

Optimizing the maintenance schedule has been widely neglected in the study of predictive maintenance, while it can be crucial for effective maintenance planning and scheduling. As an essential branch of production planning problems and the basis of maintenance scheduling optimization, the flexible job shop scheduling problem has been extensively studied in the literature (please refer to Section 5.1). According to [15], most research papers have addressed classical FJSP in the last 25 years, while only around 35% of papers considered different scenarios such as machine breakdown, uncertain processing times, overlapping operations and so on. [81] considered FJSP-PPF (process plan flexibility), where jobs can have alternative process plans. It was assumed that the process plans are known in advance and that they are represented by linear precedence relationships. Because only one of the alternative plans had to be adopted for each job, the FJSP-PPF dealt with not only routing and sequencing sub-problems, but also the process plan selection sub-problem. In this paper, a mixed-integer linear programming model was developed for the FJSP-PPF and makespan was adopted as the single performance measure.

In [32], a mathematical model and a genetic algorithm were proposed to handle the feature of overlapping in operations. It was assumed that a lot which contains a batch of identical items is transferred from one machine to the next only when all items in the lot have completed their processing, therefore, sublots are transferred from one machine to the next for processing without waiting for the entire lot to be processed at the predecessor machine, meaning that starting a successor operation of job is not necessary to finish of its predecessor completely.

Three features were considered in [129], which were (1) job priority; (2) parallel operations: some operations can be processed simultaneously; (3) sequence flexibility: the sequence of some operations can be exchanged. A mixed integer linear programming formulation (MILP) model was established to formulate the problem and an improved differential evolution algorithm was designed.

Because of unexpected events occurring in most of the real manufacturing systems, there is a new type of scheduling problem known as the dynamic scheduling problem. This type of problem considers random machine breakdowns, adding new machines, new job arrival, job cancellation, changing processing time, rush order, rework or quality problem, due date changing, etc. Corresponding works on the FJSP include [1], [2], [42], [94].

Compared with the FJSP introduced in Section 5.1.1, the VFMSO problem has some special properties: (1) flexible sequence: the sequence of the components is not predefined, and the starting time of each component is mainly determined by its due date. (2) multiple problem parameters: besides the processing time, other problem parameters like the maintenance cost, set-up time, set-up cost, repair teams, the demand for cars at a specific time, also have an impact on the maintenance schedule.

5.2.3 Customized Algorithm

A specialized evolutionary algorithm framework is defined and applied with MOEAs to solve the multi-objective vehicle fleet maintenance scheduling optimization (MOVFMSO) problem. In this part, the approach underlying the algorithm and the implementation of genetic search, including chromosome encoding, chromosome decoding and genetic operators, are described.

Components Grouping

When scheduling components, the idea of grouping several components of the same car for one visit is employed in the proposed algorithm. By grouping the maintenance of multiple components into one maintenance operation, the set-up cost and set-up time apply only once for the complete group of components. However, the maintenance cost could be indirectly penalized:

- with the reduction of the component useful life if the maintenance date is shifted backward;
- with the increase of the risk of breaking down on the road if the maintenance date is shifted forward.

Therefore, the maintenance of each component should not be shifted too far from its due date. In the proposed algorithm, an interval or execution window of the starting time is defined for each component, and the maintenance of the component can only start at a time spot inside the corresponding interval. The interval of each component consists of two parts and their respective lengths are defined as:

- Length of the interval before the due date: $0.3\times$ (Due date - Previous repair date);
- Length of the interval after the due date: $0.1\times$ (Due date - Previous repair date).

The interval is chosen relatively long so that maintenance before or after the interval hardly makes sense. With the interval, combining components can only be effective if their intervals overlap, and the starting time of the group maintenance must lie within the interval intersection.

Grouping components also means that not all the components can be maintained exactly at their due dates. For the component that is maintained before or after its due date, an extra cost is introduced to penalize either the reduction of the useful life when the maintenance is performed before the due date or the increase of component failure probability when the maintenance is performed after the due date. Figure 5.4 gives an example of the execution window and penalty function of a component. Two linear penalty functions are proposed to calculate the penalty cost based on the following assumptions.

- If a component is maintained at the same time as the previous repair time, the penalty cost would be $c+s$: the sum of its maintenance cost and the set-up cost of the car;
- If a component is maintained at the end of the interval (the latest possible repair time), the penalty cost would be $100 * c$ (c: the maintenance cost of the component). This penalty cost is the combination of all losses: the expense needed if the defect occurs on the use (diagnostics, technical and logistics support, repair or replacement of the failed component), the loss of reputation, and so on.

As can be seen, if the component O_{ij} is maintained at t_{ij} (t_{ij} is in its interval), its penalty cost is:

$$\begin{array}{ll} ((c+s)/(D_{ij}-R_{ij})) \times (D_{ij}-t_{ij}) & \text{if} \quad t_{ij} < D_{ij} \\ 0 & \text{if} \quad t_{ij} = D_{ij} \\ ((100 \times c)/((D_{ij}-R_{ij})/10)) \times (t_{ij}-D_{ij}) & \text{if} \quad t_{ij} > D_{ij}. \end{array}$$

When components are grouped together, the maintenance time of the group is the sum of the processing times of all components in the group plus one set-up time; the cost of the group is the combination of one set-up cost, the maintenance costs and the penalty costs of all components in the group.

Chromosome Encoding

EAs typically start with a diverse set of feasible solutions (a population) and iteratively replace the current population by a new population. A suitable encoding is required for the problem. Based on the properties of the application problem, a three-vector chromosome (Figure 5.5) is proposed to represent an individual, which includes:

- group structure vector: the group structures of vehicles one by one;
- starting time vector: the starting times of group operations;
- workshop assignment vector: the workshops of group operations.

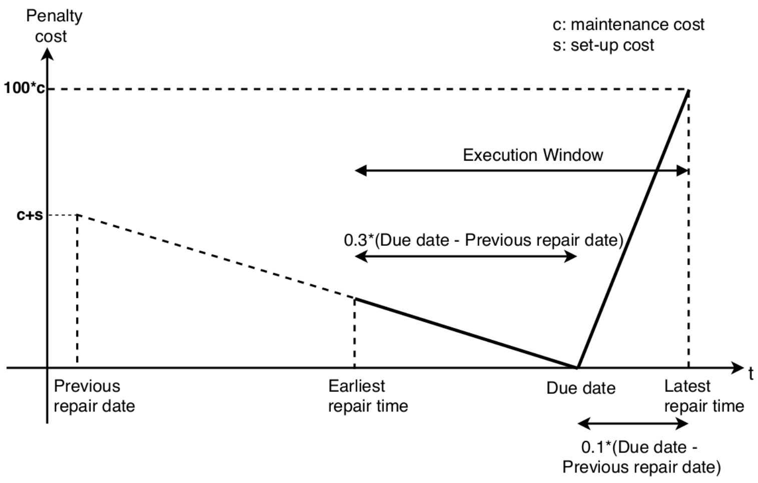

Figure 5.4: Execution window and penalty function of a component.

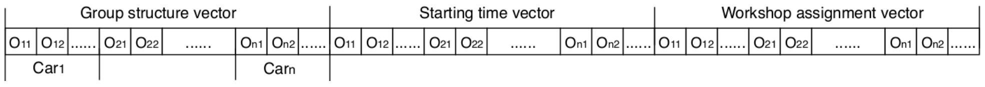
Group structure vector
Starting time vector
Workshop assignment vector
O11 O12 ...... O21 O22 ...... On1 On2 ......
Car1
Carn

Figure 5.5: Three-vector chromosome.

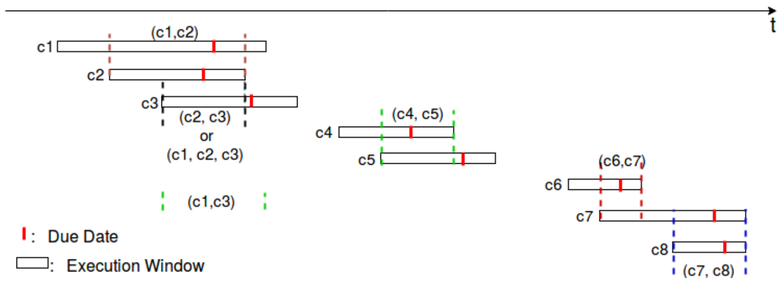

Figure 5.6: Possible groups of a car with eight components.

1. Group structure vector Figure 5.6 represents the intervals of eight components of a car. Component $c1$ can be grouped with $c2$ and/or $c3$ due to the overlap between their execution windows. Other possible group structures can be deduced in the same manner. However, the practical situation can be more complicated. For example, in Figure 5.7, $c1$ can be grouped with $c2$; $c1$ can be grouped with $c3$ and $c4$; $c1$ can also be grouped with $c3$ and $c5$. But $c1$ and $c2$ together cannot be grouped with other components; $c1$, $c3$ and $c4$ together cannot be grouped with $c5$. The overlaps and possible group structures of all components from one car are checked by the following

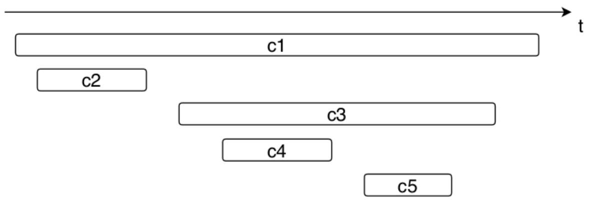

Figure 5.7: Five component intervals of a car.

sweep line algorithm:

1. All components of one car are sorted as a list according to the left end of the intervals.
2. Starting from the first component, all components are checked one by one. For each component, only this component and the subsequent components in the list are necessary in the check because the possible groups with the preceding components have been found when checking those components.
3. A stack is utilized to store the intervals and their corresponding components. The interval of the checked component is pushed onto the stack first. Look at the next component, if there is an overlap between them, push the overlap interval onto the stack; otherwise, pop off the topmost element from the stack, return the only group possibility and end the checking process.
4. Compare the topmost interval on the stack with the interval of the next component, if they overlap, push the overlap interval onto the stack; otherwise, pop off the topmost element from the stack. Repeat this step until the interval stack is empty or the last component has been checked.

All possible group numbers will be found after the above steps. Figure 5.8 shows the alternative groups of components on Figure 5.6 (left table) and Figure 5.7 (right table). Each component corresponds to a column and the numbers in its column are all the possible group numbers the component can choose. Components which have picked the same group number belong to the same group. Table 5.10 shows two randomly generated group structure vectors of the car in Figure 5.6. The first example "1 2 1 4 5 6 7 7" represents that $c1$ and $c3$ are in the same group; $c7$ and $c8$ are in the same group; $c2$, $c4$, $c5$, $c6$ are in four different groups. It does not matter what the group number is, the group number only tells us which components are in the same group. For example, "3 4 5 5 7" could be a group structure vector of the car in Figure 5.7.

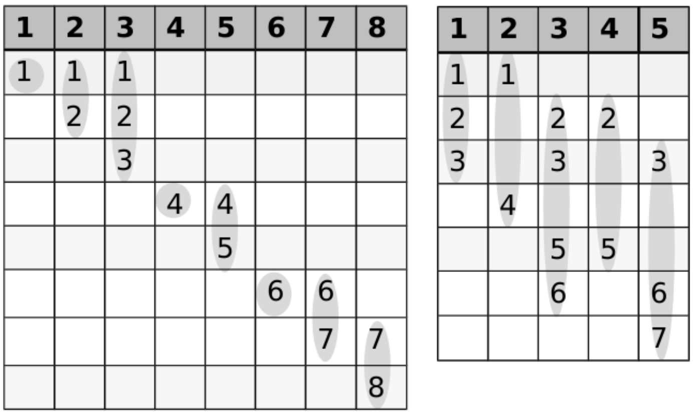

Figure 5.8: Alternative groups of components in Fig.5.6 and Fig.5.7.

Table 5.10: Examples of group structure.

Component	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Example 1	1	2	1	4	5	6	7	7
Example 2	1	1	1	4	4	6	7	8

2. Starting time vector A starting time vector can be randomly generated based on its group structure vector. Before picking the starting time for each group, the interval intersection of each group is calculated and the starting time is randomly picked from this intersection. Under the condition that a component is the sole member of a group, its starting time can be selected from its entire maintenance interval.

3. Workshop assignment vector A workshop is considered as "several workshops" based on its capacity (the number of teams). By this way, the schedule of each workshop team can be achieved from the solution. For example, consider that two workshops have three and four repairing teams respectively. Then, group operations can be randomly assigned to seven "workshops", the former three and the latter four represent corresponding teams in two workshops.

Genetic Operators

Based on the encoding, the corresponding genetic operators are designed, i.e., crossover and mutation operators. Crossover operators are applied separately to the three parts of the chromosomes.

For the group structure vector, one point or multi-point crossover can be used as crossover operator. Figure 5.9 is an example of one point crossover on the two group structure vectors in Table 5.10. Two new vectors are generated, and they are always valid because the group number of each component is from its own alternative group numbers. Because the whole group structure vector consists of all car components, one point crossover may not be efficient enough. Therefore, multi-point crossover can be applied.

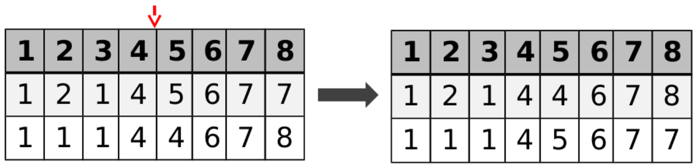

Figure 5.9: One point crossover.

The same cutting points are applied to the starting time vector when doing crossover. However, the change on the group structure vector caused by the crossover can result in the invalidity of the starting time vector because it is possible that the group members and intersection have changed due to the new group structure. Therefore, when performing the crossover on the starting time vector, the starting times of all group operations are checked and a new starting time is generated randomly from the new feasible intersection in the case that the starting time of a group is invalid.

A multi-point crossover can be applied to the workshop assignment vector as well. Every component is assigned a workshop team randomly. In the proposed algorithm, the workshop team of a group operation is decided by the first component in that group in case different workshop teams are assigned to components in the same group.

The mutation operators applied to three parts of the chromosome are also implemented separately. One or more gene values in the group structure vector can be exchanged by another alternative value from the same column in Figure 5.8 in order to generate a new individual. Again, the change applied to the group structure vector can result in the invalidity of the starting time vector. Hence, the starting time is checked for the corresponding groups after the mutation is done on the group structure vector; a valid starting time is generated randomly if it is invalid. Afterwards, some gene values can be altered in the starting time vector and workshop assignment vector to generate a new individual.

Chromosome Decoding

Decoding the chromosome is to convert an individual into a feasible schedule to calculate the objectives and constraints which represent the relative superiority of a chromosome. The group structure, the starting time and the workshop team of the operations can be obtained from each individual. Thereafter, the objective values can be calculated. When converting an individual to a schedule, it is possible that the processing times of two or more group operations assigned to the same workshop team are overlapping since the starting time of each group operation is fixed in the starting time vector. In this case, the principle of first-come-first-served is followed: the starting time and processing time of the earlier started group are not changed; the starting time of the later started group operation is not changed either; but the processing time of the later group operation is increased because the later group operation can be maintained only after the previous group operation is implemented. In the algorithm, the workshop abilities (components a workshop can maintain) are used as constraints to guarantee that a component will not be assigned to a workshop which cannot maintain it.

5.2.4 Experimental Results

The proposed evolutionary algorithm framework has been combined with four MOEAs: NSGA-III, SMS-EMOA, DI-MOEA (DI-1 and DI-2) to solve the real-world vehicle fleet maintenance schedule optimization problem, and their performance is presented in this section. Three application instances with different sizes are generated and their parameters are listed in Table 5.11. For example, the problem $P1$ includes 30 cars, each car consists of 4 components and 2 workshops are available; the two workshops have 3 and 4 repairing teams, and they can only maintain component o_1, o_2, o_3 and o_2, o_3, o_4 respectively. In order to make the results straightforward and better comparable, the processing time, cost as well as other variables are set to fixed values according to Table 5.9; the due dates and previous repair dates are generated randomly in 100 days (negative values are used for the previous repair dates); vehicle demands are randomly generated for each problem in a way that the demand never exceeds the total number of cars in the fleet.

All the experiments are performed with a population size of 100 and a budget of 500000 fitness evaluations. Both crossover and mutation probability are set to 1. For each problem, 30 optimization trials are performed with each algorithm. 10 cutting points are used in the multi-point crossover.

Table 5.11: Problem parameters.

Problem	#car	#comp	#ws	ws #car	ws #comp
$P1$	30	4	2	3	o_1, o_2, o_3
				4	o_2, o_3, o_4
$P2$	40	4	2	3	o_1, o_2, o_3
				4	o_2, o_3, o_4
$P3$	40	4	3	3	o_1, o_2, o_3
				4	o_2, o_3, o_4
				3	o_1, o_2, o_3, o_4

When analyzing the performance of different MOEAs, the empirical attainment function (EAF) is used to visualize the attained parts of the objective space. The 50% attainment surface shows that half of all Pareto optimal solutions will weakly dominate this surface and it is an estimator of what one would expect to achieve in 50% of runs [18]. Besides the average performance, the aggregate Pareto front approximation over 30 runs, i.e., the accumulated non-dominated solutions from 30 runs, has been used as another performance metric as well. Because extreme solutions are not preferable for the application problems and only one solution will be chosen to be deployed in workshops, the Pareto front approximation is zoomed in and the 50% attainment surface and the aggregate Pareto front approximation on the knee regions are plotted. Since relatively loose vehicle demands are considered currently, all the vehicle demands have been satisfied in all the solutions on the Pareto front approximations obtained from all algorithms. Therefore, the two-dimension plot can be shown to observe the results.

The performance of the algorithms has also been examined using the hypervolume indicator. Table 5.12 shows the aggregate and median hypervolume across 30 runs on three problems. For each instance, the upper row is the aggregate hypervolume, the middle row is the median hypervolume and the lower row is the standard deviation; the best hypervolume value has been highlighted in bold. When calculating the hypervolume indicator, the reference point is used by the maximum extent of the population plus an offset. It can be observed that SMS-EMOA, DI-2 and DI-1 performs best on $P1$, $P2$ and $P3$ respectively for the aggregate hypervolume; for the median hypervolume, SMS-EMOA performs best on $P1$, $P3$ and DI-1 performs best on $P2$.

For $P1$, if all the components are maintained exactly at their due dates, the total time and cost would be around 360 days and €36000, and vehicle demands cannot be guaranteed. Likewise, the total workload and cost would be around 480 days and €48000 for $P2$ and $P3$ when all the components are maintained exactly at their due

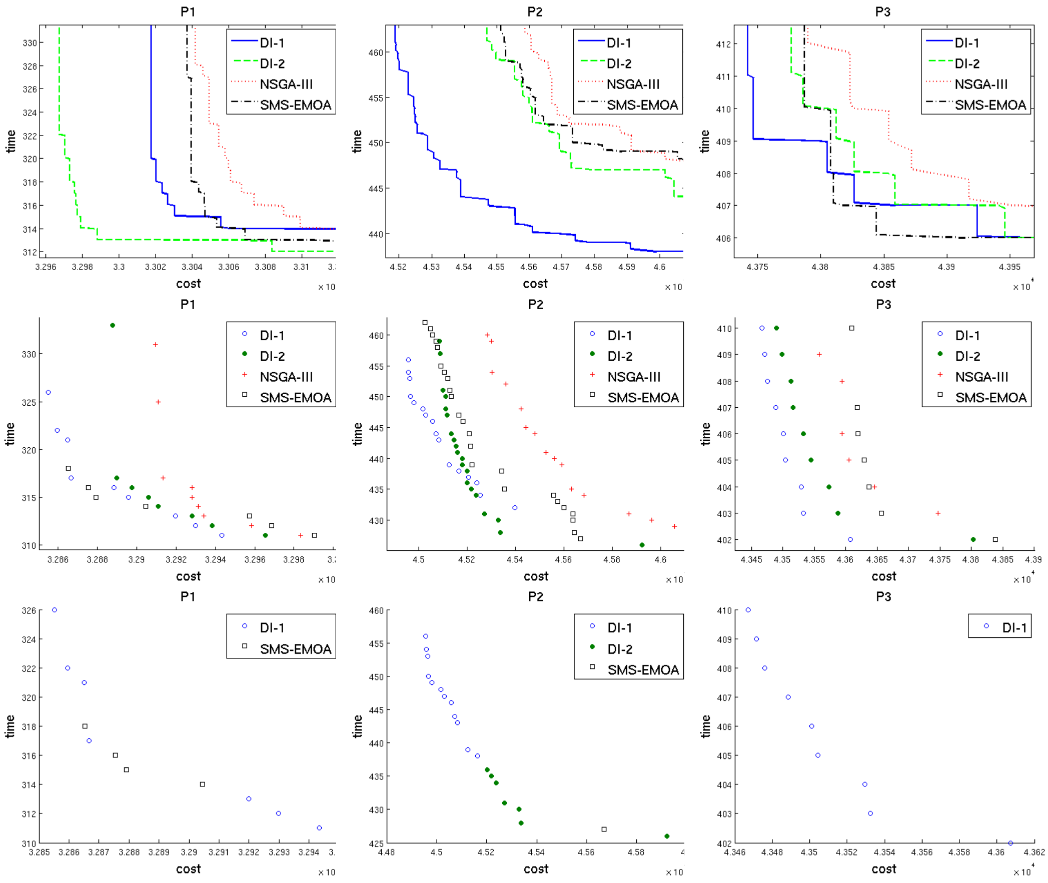

Figure 5.10: The 50% attainment surface (upper row), aggregate Pareto front approximation of each algorithm (middle row) and aggregate Pareto front approximation of all algorithms (lower row) of three instances on cost(days) & time(€).

Table 5.12: The aggregate hypervolume (Agg-HV) and median hypervolume (M-HV).

Algorithms / Problems		DI-1	DI-2	NSGA-III	SMS-EMOA
*P*1	Agg-HV	0.87983	0.91007	0.89932	**0.91633**
	M-HV	0.82728	0.83160	0.82506	**0.84137**
	std	0.0609	0.0506	0.0444	0.0563
*P*2	Agg-HV	0.89830	**0.98095**	0.93151	0.97677
	M-HV	**0.78950**	0.77907	0.72469	0.72647
	std	0.1649	0.1634	0.1259	0.1463
*P*3	Agg-HV	**0.88581**	0.88202	0.79875	0.87652
	M-HV	0.47185	0.51326	0.43115	**0.52816**
	std	0.2357	0.2244	0.2051	0.2424

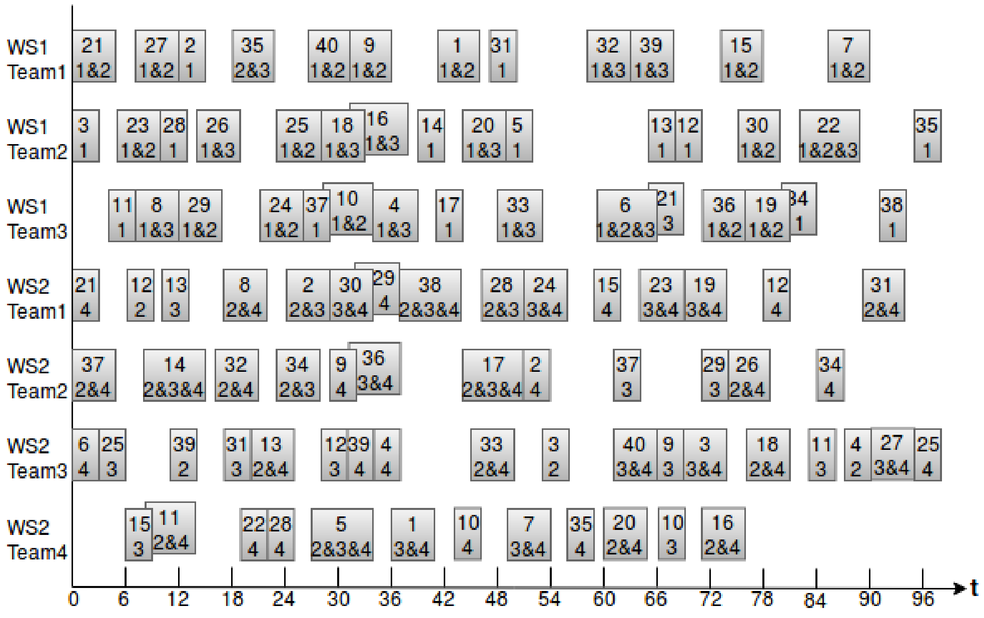

Figure 5.11: A schedule on the Pareto front approximation of DI-2 on $P2$.

dates. While, we can find many solutions with the workload less than 360 days and cost less than €36000 on the Pareto front approximations of all algorithms for $P1$. The same applies analogously for $P2$ and $P3$. Besides that the workload and cost of these solutions are already better than performing the operations on the due dates, all the vehicle demands have been satisfied at the same time.

Figure 5.11 presents a schedule on the Pareto front approximation of DI-2 on $P2$, the two objectives (cost and time) of this solution are €45338 and 428 days, the customer demands are totally satisfied. Each item is a group operation, the number above is the car number and the number below is the component number. We can see that some components are maintained individually and some components of a car are grouped together for one workshop visit. There is no maintenance of component $c4$ in workshop $ws1$ because this workshop has no ability to maintain component $c4$, and the same for component $c1$ on $ws2$. It can also be observed that some operations are overlapping to some extent. For example, the first item on $Team4$ of $ws2$ is an operation of $c3$ on car 15, its maintenance starts from day 6 and ends on day 9 because the total processing time plus set-up time is three days. However, before the completion of this task, car 11 (for the operation of $c2$ and $c4$) is sent to the same team and has to wait for one day for maintenance. The reason why car 11 waits in the workshop instead of being sent to the workshop one day later is that the failure probability of $c2$ or $c4$ would increase if this car is in use for one more day. There is no penalty cost for the wait of the car in the workshop because it has been "punished" by the increased processing time.

5.2.5 Conclusion

The real-world problem of vehicle fleet maintenance scheduling optimization is formulated. The penalty function is defined in order to deal with uncertainties in the due dates and to prevent too early or too late maintenance. A problem-specific multi-objective evolutionary algorithm framework is designed based on the component grouping strategy. State-of-the-art MOEAs are incorporated with the algorithm framework to solve the application problem and their behavior is investigated. Although DI-MOEA is used for the first time for a real-world application problem, its performance is comparable with and for some instances even better than other popular MOEAs such as NSGA-III and SMS-EMOA. DI-MOEA, especially DI-1, is the best both for the average performance and the aggregate Pareto front approximation. For the hypervolume indicator, DI-MOEA and SMS-EMOA are the best on all three instances. According to the observation of solutions, it has been found that most components are maintained earlier than their due dates in the case that their maintenance times are shifted, the reason is that the penalty costs are too expensive if they are shifted forward. In order to increase the probability that the maintenance is shifted forward, the corresponding parameter can be adjusted.

The proposed algorithm framework can work for the generic application in various similar scenarios, for example, the aircraft maintenance, ship fleet maintenance, and so on. The problem formulation and the parameters can be flexibly adjusted based on the real application or by the decision maker. In the real-world applications, it is desirable to generate schedules that are robust within a reasonable range of disruptions and uncertainties such as machine breakdowns and processing time variability. Therefore, dynamic elements will also be taken into account in the next step.