The Permutation Flow-Shop Scheduling Using a Genetic Algorithm-based Iterative Method

Mahdi Eskenasi and Mehran Mehrandezh*
Department of Engineering and Applied Science, University of Regina, Canada

Abstract

The objective is to investigate a well-known scheduling problem, namely the Permutation Flow-Shop Scheduling with the makespan as the objective function to be minimized. Various techniques, ranging from the simple constructive algorithms to the state-of-the-art techniques, such as Genetic Algorithms (GA), have been cited in the pertinent literature to solve this type of scheduling problem. A new GA-based solution methodology was developed and implemented. In this context, the performance of a stand-alone genetic algorithm (referred to as the non-hybrid genetic algorithm) and a novel hybridized genetic algorithm amalgamated with an iterative greedy algorithm were studied. The parameters of the hybrid and the non-hybrid genetic algorithms were tuned using a Full Factorial Experimental Design and Analysis of Variance. The performance of the properly tuned hybridized GA-based algorithm was examined on the existing standard benchmark problems of Taillard and it was shown that the proposed hybridized genetic algorithm performs very well on the benchmark problems.

Keywords: Flow-shop scheduling; Meta-heuristics; Genetic algorithms; Greedy search; Design of experiments; Analysis of variance

Permutation Flow-Shop Scheduling Problem (PFSP)

In assembly lines of many manufacturing companies, there are a number of operations that have to be performed on every job. Frequently, these jobs can follow the same route in assembly line meaning that the processing order of the jobs on machines should remain the same. The machines are normally set up in series, which constitute a flow-shop processing order of the jobs on machines should remain the same. The sequence change is not allowed between machines and once the sequence of jobs are scheduled on the first machine, this sequence remains unchanged on the other machines [8].

Adopting the notation of Gen and Cheng, the PFSP can be formulated as follows. Let $p(i, j)$ be the processing time for job $i$ on machine $j$, and let $\{J_1, J_2, \ldots, J_n\}$ be a job permutation. Also, let $C(J, k)$ be the completion time of job $J$ on machine $k$, then completion times for the given permutation are:

$$C(J_i, 1) = p(J_i, 1) \quad (1)$$

$$C(J_i, 1) = C(J_i, 1) + p(J_i, 1), \text{ for } i = 2, \ldots, n \quad (2)$$

$$C(J_i, k) = C(J_i, k-1) + p(J_i, k), \text{ for } k = 2, \ldots, m \quad (3)$$

$$C(J_i, k) = \max(C(J_i, k), C(J_{i'}, k-1)) + p(J_i, k), \text{ for } i = 2, \ldots, n; \text{ for } k = 2, \ldots, m \quad (4)$$

The makespan for the given permutation is obtained through:

---

*Corresponding author: Mehrandezh M, Department of Engineering and Applied Science, University of Regina, Canada, Tel: 306-585-4658; Fax: 306-585-4855; E-mail: mehran.mehrandezh@uregina.ca, mehranmehrandezh@gmail.com

Received November 24, 2015; Accepted June 24, 2016; Published June 30, 2016


Copyright: © 2016 Eskenasi M, et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.
The permutation flow-shop scheduling problem as presented above is a combinatorial optimization problem and the size of its search space is \( n! \). According to Reeves and Yamada [9] any improvement in the quality of the solution (i.e., makespan) obtained for the general flow-shop scheduling is rather small as compared to that of the permutation flow-shop scheduling. On the other hand, the general flow-shop scheduling causes the size of search space to increase considerably from \( n! \) to \((n!)^m\). Therefore, in the general form of the flow-shop scheduling (where the permutation of jobs is allowed to be different on each machine) one might obtain a slightly better makespan at the cost of much higher computation time.

The layout of this paper is as follows: A review of the literature is provided in Section 2. The most celebrated constructive method, namely NEH is explained in Section 3. A review of the Genetic Algorithm, as the base for our proposed solution, is given in Section 4. Chapter 5 and 6 explain about the proposed solution methodology, numerical results, and the statistical analysis of the results. Conclusions and future work are provided in section 7.

Review of Literature

The methods cited in the literature for the PFSP can be divided into two main categories; namely, exact methods and heuristic methods. French [10] refers to exact methods as enumerative algorithms. Enumerative algorithms such as integer programming, and branch and bound techniques can be theoretically applied to find the optimal solution for the permutation flow-shop scheduling problem (See, e.g., French [10], Ladhari [11]). However, these methods are not practical for large-sized or even medium-sized problems. Ruiz et al. [5] states that exact methods are only applicable for small instances when the number of jobs is less than 20. Therefore, researchers have focused on applying heuristic methods to the PFSP in order to find near-optimal solutions in much shorter time. The heuristics proposed for the PFSP can fall into either constructive or improvement heuristics.

According to French [10], a constructive heuristic for a scheduling problem can be defined as an algorithm that builds up a schedule from the given data of the problem by following a simple set of rules (e.g., First-In-First-Out) which exactly determines the processing order.

Improvement heuristics, as contrasted to constructive heuristics, start from a previously generated schedule and try to iteratively modify it. Meta-heuristics (modern heuristics) such as Genetic Algorithms, Simulated Annealing, Iterated Local Search, Iterated Greedy Search, etc. fall in the category of improvement heuristics.

An extensive literature review on the proposed heuristics for the PFSP can be found in Gen and Cheng, and Ruiz and Maroto [6]. The heuristics by Johnson [12], Campbel et al. [13], Dannenbring [14], Palmer [15], Gupta [16], Nawaz et al. [17], Taillard [18], Hundal and Rajgopal [19], Koukamas [20], and Pour [21] are instances of constructive heuristics in the literature. The most cited existing improvement heuristics in the literature are Simulated Annealing (SA) of Osman and Potts (1989), SA of Ogbu and Smith [22], Genetic Algorithm (GA) of Chen et al. [23], GA of Reeves [2], GA of Murata et al. [24], Iterated Local Search of Stützle [3], GA of Ponnambalam et al. [25], GA of Iyer and Saxena [4], GA of Ruiz et al. [5], Iterated Greedy Search of Ruiz and Stützle [7].

The Algorithm of Nawaz, Enscore and Ham [17]

The algorithm of Nawaz et al. [17], commonly referred to as the NEH algorithm in the literature, has been given considerable attention in the literature to the PFSP. It has been unanimously referred to as the best constructive method for the permutation flow-shop scheduling problems among the constructive heuristics in the literature; (See, e.g., Turner and Booth [26], and Widmer and Hertz [27], Taillard [18], Ponnambalam et al. [25], Ruiz and Maroto [6]). To the best of our knowledge, among the aforementioned references, the work of Ruiz and Maroto [6] is the most comprehensive since they have compared 15 constructive heuristics extensively.

In the NEH algorithm, first the jobs are sorted by decreasing sum of processing times on machines. Then the first two jobs with highest sum of processing times on the machines are considered for partial scheduling. The best partial schedule of those two jobs (i.e., one that produces lower partial makespan) is selected. This partial sequence is fixed in a sense that the relative order of those two jobs will not change until the end of the procedure. In the next step, the job with the third highest sum of processing times is selected and three possible partial schedules are generated through placing the third chosen job at the beginning, in the middle, and at the end of the fixed partial sequence. These three partial schedules are examined and one that produces minimum partial makespan is chosen. This procedure is repeated until all jobs are fixed and the complete schedule is generated.

Taillard [18] investigated some of the above-mentioned constructive algorithms and he showed that the performance of the NEH algorithm is superior in terms of the achieved makespan through some experimental results. He proposed a method to speed up the NEH algorithm, which is known as Taillard’s algorithm [18]. One should note that Taillard’s algorithm [18] achieves the very same results that the NEH algorithm does, in lower CPU time.

Genetic Algorithms and Their Applications in the PFSP

Genetic algorithms are stochastic search and optimization techniques developed based on the mechanism of Darwinian ‘natural evolution’ and ‘survival of the fittest’. The probabilistic search method in Genetic Algorithms (GAs), along with natural selection and reproduction methods mimic the process of biological evolution [28].

A GA is a probabilistic algorithm that maintains a population of individuals. Each individual in population is referred to as a chromosome representing a potential solution (i.e., fit) to the problem at hand. A chromosome is encoded into a string of symbols, which might have a simple or complex data structure.

The collection of individuals (chromosomes) forms the population, and the population evolves iteration by iteration. Each iteration of the algorithm is named a generation. In each generation, every chromosome is evaluated and is given some measure of fitness. Then, a new population is generated by selecting the more fit individuals. Some of the individuals of the new population are transformed through genetic operators. There are mainly two types of transformation; namely mutation (unary operator) and crossover (binary operator). In mutation, a single individual is usually slightly changed and added to the population. In crossover, a new individual is generated through exchanging the information between two individuals. These selection, alteration, and evaluation steps are repeated for a pre-set number of generations. At the final generation, the algorithm converges to the best chromosome, which hopefully represents a near-optimum solution to the problem (with the highest fit value) [29].

Representation and initial population

As mentioned earlier, in the GAs each individual, as a potential
solution, is encoded into a string of symbols. In the classical GAs proposed by Holland [28], the binary representation of individuals has been applied. This, however, does not suit the representation of schedules in a PFSP. In practice, the binary representation for the PFSP needs special repair algorithms, because the change of a single bit of the binary string from zero to one (or vice versa) may produce an illegal schedule (a schedule which is not a permutation of jobs). Instead, the most frequently applied encoding scheme, and perhaps the most ‘natural’ representation of a schedule for the PFSP, is a permutation of the jobs. A permutation of jobs determines the relative order that the jobs should be performed on machines.

Initial population of a GA can be a collection of random permutation of jobs, collection of permutations which have already been obtained through heuristic approaches, and/or a combination of these two methods. Obviously, high-quality solutions generated by some heuristic method(s) might help a genetic algorithm find near-optimal solutions more efficiently. However, on the flip side one should note that starting the GA with a homogeneous population might result in a premature convergence [6].

Evaluation

As previously stated, the makespan in a PFSP is considered as the objective function to be minimized. Consequently, the fitness value of an individual should be calculated based on its makespan. Previously, the way one can obtain the makespan of a given permutation of jobs in a PFSP was formulated and illustrated.

Selection

Selection operator provides the driving force in a genetic algorithm. From biological point of view, Darwinian natural selection results in survival of the fittest. Similarly, in the selection phase of the genetic algorithms the fitter is an individual, the higher is the chance of being selected for the next generation.

It should be noted, however, that an appropriate selection method gives at least some chance of competing for survival to the individuals having lower fitness values. Otherwise, a few super individuals (highly fit individuals) will dominate the population after a few generations, and GA will terminate prematurely. Two selection methods; namely, rank-based selection [30] and tournament selection [28] are investigated in this paper.

Crossover operators in GA-based PFSP

Crossover operator is devised in GAs to exchange information between randomly selected parents with the aim of producing better offspring and exploring the search space [4]. Crossover operators mimic the reproduction mechanism in nature.

One should note that where a permutation of jobs in a PFSP represents a chromosome in the population, classical crossover operators, which are suitable for binary representation, cannot be applied. The reason is that the classical crossover operators will often produce an offspring, which is not a permissible permutation of jobs. One should also note that in an offspring each job must be represented only one time. Otherwise, the produced offspring is named illegal (inadmissible). Therefore, a customized crossover operator is needed for the problem at hand.

The most popular crossover operators applied for the PFSP are Order Crossover [31], Cycle Crossover [32], Partially-Mapped Crossover [33], Position-Based Crossover [34], Order-Based Crossover [34], Precedence Preservation Crossover [35], Two-Point Crossover I [24], Two-Point Crossover II [24], One-Point Crossover [24], Longest Common Subsequence Crossover [4], Similar Job Order Crossover [5], Similar Block Order Crossover [6], Similar Job 2-Point Order Crossover [7], and Similar Block 2-Point Order Crossover [7].

Ruiz et al. [5] have investigated the performance of the above-mentioned crossover operators except for the Longest Common Subsequence Crossover (LCSX). In their survey, it is shown that the Similar Block Order Crossover (SBOX) outperforms other crossover operators (excluding the LCSX, which was not considered in their survey).

Iyer and Saxena [4] proposed the LCSX and showed that LCSX outperform One-Point Crossover (OPX) through experimental results. However, in their experiments the standard benchmarks of Taillard were not used. In fact, the performance of their algorithm was only tested on some randomly generated problem instances of the PFPS. Consequently, the comparison of LCSX and SBOX on standard benchmarks of Taillard [18] seems necessary. In the following sections, the LCSX and SBOX are described. In Section 4, the performance of LCSX and SBOX utilized in the proposed hybrid GA is extensively studied and compared.

Longest common subsequence crossover (LCSX): The LCSX preserves the longest relative orders of the jobs which are common in both parents [4]. This concept is illustrated via an example. In the following example, two parents (schedules), P1 and P2, mate to produce two offspring, O1 and O2, via the LCSX:

\[
P1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \\
P2 = (4 \ 7 \ 6 \ 2 \ 8 \ 9 \ 1 \ 5 \ 3)
\]

Step 1: First, the longest common subsequence of the jobs is found in the parents. In this particular example, one can find several common subsequences. For instance (4,7) is a common subsequence between P1 and P2, since the relative order of these two jobs in both parents remains the same. Another common subsequence between P1 and P2 is (4,7,8). After a simple investigation one can readily realize that the longest common subsequence between P1 and P2 is (4,7,8,9).

The jobs belonging to the longest common subsequence are underlined in parents.

\[
P1 = (1 \ 2 \ 3 \underline{4} \ 5 \ 6 \underline{7} \underline{8} \underline{9}) \\
P2 = (4 \underline{7} \underline{6} \ 2 \underline{8} \underline{9} \ 1 \ 5 \ 3)
\]

Step 2: The underlined jobs are directly copied from P1 into the O1 by preserving their positions in P1. Likewise, the underlined jobs in P2 are directly copied into the O2 (‘x’s in the offspring O1 and O2 represent unknown jobs (open positions) at the current step):

\[
O1 = (\underline{x \ x \ x} \underline{4} \underline{x \ x} \underline{7} \underline{8} \underline{9}) \\
O2 = (\underline{4} \underline{7} \underline{6} \underline{x \ x} \underline{8} \underline{9} \underline{x \ x})
\]

Step 3: The jobs in P2 which are not underlined (i.e., the jobs in P2 which are not in longest common subsequence) will fill the open positions of O1 from left to right. One should note that the relative order of jobs which are not in longest common in P2 is preserved as the open positions of O1 are being filled. Likewise, the open positions of O2 will be filled.

\[
O1 = (6 \ 2 \ 1 \ 4 \ 5 \ 3 \ 7 \ 8 \ 9) \\
O2 = (4 \ 7 \underline{6} \ 2 \underline{8} \underline{9} \ 3 \ 5 \ 6)
\]
Iyer and Saxena [4] proposed LCSX since they assume that the relative positions of the jobs constitute the building block\(^1\) of the parent chromosomes. Consequently, they proposed the crossover operator that maintains the locations of those jobs whose relative positions in both parents are identical. More specifically, LCSX preserves only the longest common subsequence in the parents, which is ‘expected’ to give good results [4]. Since the performance of LCSX has not been examined on Taillard’s benchmark problem instances [18], it is of interest to verify the conjecture of Iyer and Saxena [4] in this paper.

**Similar block order crossover (SBOX):** In the SBOX [5], similar block of the jobs occupying the same positions play the main role in the crossover process. Considering \( P_1 \) and \( P_2 \) as the parents, the SBOX is exemplified as follows:

\[
P_1 = (1 2 3 4 5 6 7 8 9) \\
P_2 = (5 3 4 6 7 8 2 1 1 1 1)
\]

Step 1: similar block of jobs are to be found in the parents. Similar blocks are (at least) two consecutive identical jobs which occupy the very same positions in both parents. (Similar blocks are underlined in this example):

\[
P_1 = (1 \underline{2} \underline{3} \underline{4} 5 \underline{6} \underline{7} \underline{8} 9) \ \\ P_2 = (\underline{5} \underline{3} \underline{4} 6 \underline{7} 8 2 \underline{1} 1 1 1)
\]

Step 2: similar block of the jobs are copied over to both offspring.

\[
P_1' = (x \underline{3} \underline{4} x \underline{6} \underline{7} 8 \ x x) \\
P_2' = (x \underline{3} \underline{4} x \underline{6} \underline{7} 8 \ x x)
\]

Step 3: All jobs of each parent up to a randomly chosen cut point (marked by \( '|' \) are copied to its corresponding offspring.

\[
O_1' = (1 x \underline{2} \underline{3} \underline{4} \underline{6} \underline{7} 8 \ x x) \\
O_2' = (2 \underline{5} \underline{3} \underline{4} x \underline{6} \underline{7} 8 \ x x)
\]

Step 4: Finally, the missing jobs of each offspring are copied in the relative order of the other parent.

\[
O_1 = (1 \underline{2} \underline{3} \underline{4} \underline{5} \underline{6} \underline{7} 8 9 11) \\
O_2 = (2 \underline{5} \underline{3} \underline{4} 1 \underline{6} \underline{7} 8 9 10 11)
\]

Ruiz et al. [5] proposed SBOX because they conjectured that preserving similar blocks of the jobs (defined earlier) plays the role of the building block in the chromosomes of the parents and consequently these blocks must be passed over from parents to offspring unaltered. The SBOX conceptually complies with the Johnson’s [12] rule for scheduling jobs, namely, job \( 1 \) proceeds job \( j \) if \( \min (P_{ij}, P_{j}) < \min (P_{i}, P_{j}) \) whereas, \( P_{ij} \) is the processing time of job \( "x" \) on machine \( "y" \).

**Mutation operators**

In order to keep a GA form focusing on one region of the search space and from ending up with local optima, mutation operation has been introduced in the GAs. One should note that in the proposed genetic algorithms for the PFSP, mutation probability is normally assigned to each individual whereas in classical binary representation of genetic algorithms, mutation probability is usually applied to each bit (i.e., each position of the string) and not to an individual. In the following section the insertion mutation will be described.

---

\(^1\)According to Haupt and Haupt (2004), building block can be defined as the patterns that give a chromosome a high fitness and increase in number as the GA progresses. Also, see schema theorem in Michalewicz [29].
For producing \( \text{POP}_{\text{size}} \) permutations (\( \text{POP}_{\text{size}} \) represents the size of population) as the initial population of the GA, the following strategy is adopted: The first permutation of jobs (as the first individual in the initial population) is generated through the original NEH algorithm. For initializing the rest of the population, which includes \( \text{POP}_{\text{size}−1} \) permutations of jobs, \( \text{POP}_{\text{size}−1} \) random permutations are sent to the modified NEH, and, finally, \( \text{POP}_{\text{size}−1} \) permutations of jobs are randomly chosen (with the probability of \( p_m \)) to undergo crossover operation in the next step. Each pair which undergoes crossover operation produces two offspring that replace the parents in the candidacy list. If a pair is not chosen for crossover, they will remain intact in the candidacy list. Consequently, after applying the crossover, the number of individuals in the candidacy list remains unchanged. LCSX and SBOX, described are examined as the crossover operators.

After applying the crossover operators, a fraction of individuals in the candidacy list is randomly chosen to undergo mutation operator. More specifically, the mutation operator with the probability of \( p_m \) is applied to each individual of the candidacy list. Two mutation operators are examined in the proposed algorithm; namely insertion mutation and destruction-construction mutation. The destruction-construction mutation operation is the same as destruction and construction phases of the Iterated Greedy Algorithm, which will be discussed later on.

The preliminary tests show that destruction-construction mutation adopted in this paper is superior to the insertion mutation. Therefore, insertion mutation will not be considered in design of experiments (to be discussed in Section 6). Three levels are considered for mutation rates, namely, 0.10, 0.15, and 0.20.

**Termination condition**

A time dependent termination condition of \( n \times m \times 90 \) milliseconds is adopted, where \( n \) and \( m \) are the number of jobs and machines, respectively. The proposed GA is implemented in Java (Java 2 Platform, Standard Edition) and is run on a personal computer with 1.4 GHz Pentium III processor, and 2 GB of RAM. The aforementioned period (as the termination condition) is twice the duration that Ruiz et al. [5] used in their proposed GA. In fact, Ruiz et al. [5] employed faster computer\(^{1}\) with Athlon XP 1600+ processor (running at 1400 MHz) and 512 MB of RAM [5]. The hybrid method based on the GA and the IGA is addressed in the next sections. An overview of the IGA is given first, and then the proposed hybrid solution technique is addressed.

**Iterated greedy algorithm**

Ruiz et al. [6] have proposed an Iterated Greedy Algorithm for the PFSP recently, which is easy to implement. The central idea behind any Iterated Greedy Algorithm (IGA) is to iteratively apply two main stages to the current solution (e.g., permutation, etc.); namely destruction stage and construction stage. These two stages are problem specific and in the case of the PFSP they are tuned by Ruiz et al. [6] as follows: in destruction stage, some jobs are removed at random from the current solution, and in construction stage, those removed jobs are added again to the sequence in a certain pattern which will be described later on.

The Iterated Greedy algorithm (IGA) of Ruiz and Stützle [6], which is similar to the Iterated Local Search (ILS) of Stützle [3] to some extent, has five components, namely, initial solution generation, destruction, construction, local search, and acceptance criterion.

The initial solution is found through the NEH heuristics which is followed by a local search. Local search, which will be described shortly, explores and finds local optimal solution in the neighbourhood of the initial solution. After finding the local optimal solution, four components, namely destruction, construction, local search, and acceptance criterion are iteratively applied to the incumbent solution.

In destruction phase, \( d \) jobs (without repetition) are randomly removed from the current solution \( \pi \). Removing \( d \) jobs from a permutation of \( n \) jobs, yields two partial sequences; namely \( \pi_d \) which

\[ 1) \text{Generate a random permutation of } n \text{ jobs.} \\
2) \text{Take the first two jobs and schedule them in order to minimize the partial makespan as if there were only these two jobs.} \\
3) \text{For } k = 3 \text{ to } n \text{ do: Insert the } k\text{-th job at the place, among the } k \text{ possible ones, which minimizes the partial makespan.} \\

Figure 1: The proposed modified NEH algorithm as the initial population generator.

---

\(^{1}\)One should admit that it is not easy to compare the performance of these two different types of computers precisely.
contains \((n-d)\) jobs, and \(\pi_i\) which has \(d\) jobs. Needle to say, the order in which those \(d\) jobs are removed from \(\pi\) is important because this order provides the \(\pi_{ir}\), which is one of the two mentioned partial schedules.

In construction phase the jobs of \(\pi_i\) are re-inserted to \(\pi_{1}\) one by one according to the 3\textsuperscript{rd} step of the NEH algorithm. Thus, the first job of \(\pi_i\) has \((n-d+1)\) possible positions for insertion to \(\pi_{ir}\). All these positions are examined and the position which yields the best makespan is chosen. After fixing the position of the first job of \(\pi_{ir}\), the best position for the second job of \(\pi_{ir}\) is investigated. The second job of \(\pi_{ir}\) has \((n-d+2)\) possible positions for insertion to \(\pi_{ir}\). This procedure is repeated until the last job of \(\pi_{ir}\) is inserted to \(\pi_{1}\). One should note that Taillard’s acceleration is applied in construction phase.

After the construction phase, the current solution can undergo local search (this can be optional). In fact, Ruiz et al. [7] realized that this step improves the overall performance of the IGA. The insertion-move, which is similar to insertion mutation operator, is applied as a local search technique. As mentioned earlier, \((n-1)^2\) possible neighbourhoods can be scanned for a given permutation. Ruiz et al. [7] follow the strategy in which the first randomly chosen neighbourhood (among \((n-1)^2\) possible neighbourhoods) that improves the makespan of current solution is applied as the output of the local search procedure. In this paper, however, a different approach is adopted. More specifically, all \((n-1)^2\) possible neighbourhoods of the current solution are scanned and the best one is applied as the output of the local procedure.

The acceptance criterion that is applied in the IGA is a simulated annealing-type acceptance criterion, and is similar to the acceptance criterion in ILS [38]. Suppose that \(\pi\) is the current solution and after \(\pi\) undergoes local search, \(\pi^*\) is yielded. If \(C_{\text{max}}(\pi^*)\) is worse (i.e., greater) than \(C_{\text{max}}(\pi)\), \(\pi^*\) is accepted as a current solution with the probability of:

\[
p = \exp\left(-\frac{(C_{\text{max}}(\pi^*) - C_{\text{max}}(\pi))}{T}\right) 0 \leq p \leq 1 \tag{6}
\]

If \(C_{\text{max}}(\pi^*)\) is better than \(C_{\text{max}}(\pi)\), \(\pi^*\) is then accepted. Two parameters of IGA, namely \(T\) and \(d\) (where \(T\) and \(d\) represent the Temperature and the number of jobs to be removed in the destruction phase, respectively), were tuned by Ruiz et al. [6] through the design of experiments. As a result of experimental analysis, they found that the IGA with \(d=4\) and \(T=0.4\) yields the best results in terms of the relative percentage deviation from best-known solution of Taillard’s instances.

Before dealing with the hybridization of the GA, it should be pointed out that the mutation operation considered for the proposed GA consists of two phases, namely destruction and construction phases of the IGA with \(d=4\).

### Hybridization with iterated greedy algorithm

The idea of hybridization of the GA with other heuristics has been investigated for solving the PFSP in the literature. Murata et al. [24] examined genetic local search and genetic simulated annealing as two hybridization of genetic algorithm [24]. They showed that genetic local search outperforms the non-hybrid genetic algorithms and genetic simulated annealing. In their genetic local search, the local search is added as an improvement phase. In that improvement phase, all individuals of the current population of the GA iteratively undergo local search before going through the selection phase. The disadvantage of their approach is that the local search becomes very time consuming. Similarly, Ruiz et al. [6] showed that the hybridization of their proposed GA with a local search yields better results.

The approach that is adopted in this paper is as follows: after applying the crossover and mutation operators, Iterated Greedy Algorithm is applied to the best individual of the current population with probability of \(P_{\text{iga}}\). The best permutation of the GA is then sent to the IGA, and the IGA is run for \(n \times m \times 30\) millisecond. If the permutation, which is achieved through the IGA, has lower makespan than that of the best permutation of GA, it is replaced with the best permutation of GA. The final pseudo code for the hybrid method is illustrated in Figure 2. The levels that are considered for \(P_{\text{iga}}\) are 0.00, 0.005, 0.01, and 0.02. One should note that the case of \(P_{\text{iga}}=0\) corresponds to the non-hybrid genetic algorithm.

### Experimental Parameter Tuning of the Proposed Algorithm

In this section, the impact of the factors described in Section 5 on the performance of the proposed GA-based algorithms to solve the PFSP is investigated through full factorial experimental design and ANOVA (Analysis of Variance). The Regression Point Displacement (RPD) is used as the performance measure for this purpose. In the Design of Experiments (DOE), it is of interest to determine which factors affect the proposed algorithms (both hybrid and non-hybrid GA) the most. Moreover, the performances of the hybrid and non-hybrid GA are extensively compared, and it is shown through experimental results that the hybridization of the GA with the IGA (i.e., the hybrid GA) yields better results in terms of the RPD. It is also shown that the proposed hybrid GA is robust\(^{13}\).

#### Implementation of the full factorial experimental design

In the full factorial experimental design (Montgomery 2000), all possible combinations of the levels (treatments\(^{14}\)) of the factors that are anticipated to be influential to the end results are investigated. A combination of the following factors and their associated RPDs is referred to as observations here:

- Population size: 20, 40, and 60;
- Selection type\(^1\): tournament and rank-based selections,
- Crossover type\(^2\): LCSX and SBOX,
- Crossover probability: 0.4, 0.6, and 0.8,
- Mutation probability: 0.1, 0.15, and 0.20,
- \(P_{\text{iga}}\) (IGA Probability): 0.005, 0.010, and 0.020.

One should note that the population size, selection type, crossover probability, mutation probability, and IGA probability are controllable factors in the full factorial experimental design. In addition to those controllable factors, there are two uncontrollable factors in the DOE, namely the number of jobs (i.e., \(n\)) and the number of machines (i.e., \(m\)). As a result, eight factors altogether are examined in the full factorial experimental design.

For tuning of the proposed hybrid algorithm, the set of Taillard’s benchmarks was used. Taillard’s benchmark problems can be found in [39]. The preliminary results show that any treatment combinations of the proposed hybrid algorithm find the optimal solution for almost all small instances (namely, the instances with the sizes of \(20 \times 5\), \(20 \times 10\), \(20 \times 20\) and \(50 \times 5\), which are labeled as ta001 to ta040 in the literature).

\(^{13}\)According to Montgomery (2000), the term robust algorithm is applied to an algorithm, which is influenced very slightly by the external source of variability.

\(^{14}\)In the design of experiments, every level of a factor is called a treatment.
Therefore, in the design of experiments, these instances are not used. In addition, instances with the size of 100 × 5 (labeled as ta041 to ta070) are not considered in the design of experiment for the same reason.

The instances used in the design of experiment are as follows: five out of ten instances with the size of 50 × 10 (namely, ta041, ta043, ta045, ta047, and ta049), five out of ten instances with the size of 50 × 20 (namely, ta051, ta053, ta055, ta057, and ta059), five out of ten instances with the size of 100 × 10 (namely, ta071, ta073, ta075, ta077, and ta079), five out of ten instances with the size of 100 × 20 (namely, ta081, ta083, ta085, ta087, and ta089), five out of ten instances with the size of 200 × 10 (namely, ta091, ta093, ta095, ta097, and ta099), and finally five out of ten instances with the size of 200 × 20 (namely, ta101, ta103, ta105, ta107, and ta109).

Noticeably, 324 combinations of the controllable factors should be employed for each of 30 above-mentioned instances. Consequently, 9,720 observations are needed to conduct all possible combinations of the factors for 30 chosen instances. Since four replicates are carried out for the hybrid GA, 38,880 observations are totally needed. In short, for the implementation of full factorial experimental design, the effects of these 38,880 observations were investigated on their corresponding RPDs as the response variable.

Analysis of variance for the proposed hybrid algorithm

The full factorial design with aforementioned characteristics was conducted in MINITAB Statistical Software. MINITAB generates an Analysis of Variance (ANOVA) Table as the output of factorial design. The effects of each of the eight above-mentioned factors were individually investigated using hypotheses tests. More specifically, all possible interactions of two factors, and all possible interactions of three factors are examined using hypotheses tests.

Adopting the results of the ANOVA is subjected to satisfaction of certain assumptions, which are called the model adequacy checking [40]. The model adequacy checking can be performed through the examination of residuals, which consists of three tests; namely, test of normality of residuals, test of residuals versus the factors (the residuals should be unrelated to any factor including the predicted response variable), and test of residuals versus observation order (the residuals should be run-independent). Validation of these assumptions is investigated through graphical residual analysis, which is generated by MINITAB software. All these three assumptions were satisfied.

In order to recognize whether each factor is influential on the proposed hybrid algorithm, the associated P-value of ANOVA is employed. P-value can be defined as the smallest level of significance that would lead to rejection of null hypothesis (rejection of null hypothesis means that there is a statistically significant difference between the levels of the factor or interaction considered). The predetermined level of significance of 0.05 is normally applied in hypothesis testing, which is adopted in this paper as well [41].

The P-values for factors n, m, SelectionType, and MutationProbability are smaller than 0.05 (the predetermined level of significance). Also, the P-values for some interactions, namely (n × m), (n × CrossoverType × SelectionType) and (n × m × SelectionType) are less than 0.05. Therefore, the null hypotheses are rejected for these terms, indicating that factors n, m, SelectionType, and MutationProbability are influential on RPD. It is noteworthy that n and m are uncontrollable factors and they are directly related to the problem at hand [42]. Furthermore, the three aforementioned terms, namely (n × m), (n × CrossoverType × SelectionType) and (n × m × SelectionType) are uncontrollable since they contain at least one of the two uncontrollable factors. The impact of SelectionType and MutationProbability will be investigated on the response variable later on.

As for the remaining factors and their interactions, the corresponding P-values were greater than 0.05, indicating that the null hypothesis should be accepted for them. The statistical interpretation behind acceptance of null hypothesis is that the obtained observations do not give enough evidence in support of significant difference in the average RPD across the levels of those factors and their interactions [43]. Therefore, it can be concluded that the factors or terms that have P-values of greater than 0.05 are not influential on the overall performance of the proposed hybrid genetic algorithm.

SelectionType and MutationProbability were tuned by making use of the plots of main effects and their interactions. The procedure for tuning these two parameters is as follows: between these two factors, SelectionType (the factor that has lower P-value) is tuned first. In fact,
the lower the P-value is, the more impact that factor can have on the response variable.

The plots for main effects of SelectionType and MutationProbability are shown in Figure 3. The difference between the mean of RPD for SelectionType 1 and SelectionType 2 is greater than the difference between the mean of RPD for MutationProbability of 0.10 and MutationProbability of 0.20. In other words, this plot confirms the fact that SelectionType has more influence on the response variable as compared to the factor MutationProbability.

In Figure 4, it can be observed that the SelectionType of tournament selection (denoted by 1) is more effective as compared to the rank-based selection (denoted by 2). Considering the SelectionType of tournament for the algorithm, it is also of interest to determine what mutation probability is more effective for the algorithm. It can be seen form Figure 4 that the mutation probability of 0.1 is more effective on the response variable.

As mentioned in the previous section, the other factors and their interactions were not influential on the RPD. However, through Figure 5 it can be seen that the following remaining parameters give a slightly better results: crossover type 2 (i.e., SBOX), population size of 40, crossover probability of 0.60, and IGA probability of 0.02.

The hybrid GA was tuned with the above-mentioned set of parameters and was run ten times against each of 110 instances of Taillard (labeled ta001 to ta110). Since these 110 instances contain 11 distinct problem sizes (i.e., for each problem size ten instances are available) the results are averaged over problem sizes and are given in Table 1.

Tuning the proposed non-hybrid GA

In this section, the case of $P_{IGA}=0$ is individually investigated (i.e., the non-hybrid GA). In order to tune the parameters of the non-hybrid GA, the full factorial experimental design for all possible combinations of the following factors was carried out: population size (20, 40, and 60), selection type (rank-based selection, and tournament), crossover type (SBOX, and LCSX), crossover probability (0.4, 0.6, and 0.8), and mutation probability (0.1, 0.15, and 0.20). $P_{IGA}$ was set to zero, therefore 108 combinations were investigated.

The same thirty instances of Taillard, which were employed in implementation of the full factorial experimental design of the hybrid algorithm, were used here. Three replica with the above-mentioned combination of parameters were considered and consequently 9,720 observations were carried out. Similarly, the analysis of variance, and the model adequacy checking were investigated. The $P$-values of the ANOVA table for the non-hybrid GA were zero for all factors, meaning that all factors are statistically influential on the response variable. Since all $P$-values are zero, $P$-values are employed in order to determine the order of importance of the factors in the non-hybrid GA. The higher the $P$-value is, the more influential that factor is on the response variable. Therefore, it can be observed that the order of importance of the controllable factors in the non-hybrid GA is as follows: CrossoverType, SelectionType, PopulationSize, CrossoverProbability, and MutationProbability.

The plots of main effects (the main factors) and their interaction were employed in order to find the parameters that produce the lowest RPD. Based on the Figures 6 and 7, it is concluded that the following set of parameters is the optimum combination for the non-hybrid GA: crossover type 2 (i.e., SBOX), selection type 2 (i.e., rank-based selection), population size of 60, crossover probability of 0.40 and mutation probability of 0.20.

The mean value of RPD for the non-hybrid GA can be observed through the horizontal line illustrated in Figure 6. As it can be observed, the mean value of RPD for the non-hybrid GA is greater than 5.5 (its exact value is 5.651). The obtained mean of RPD for the hybrid GA is 0.938 (see the horizontal line illustrated in Figure 5). Consequently, the mean value of RPD for the non-hybrid GA was significantly higher than that of the hybrid GA. In other words, the hybridization of the GA with the IGA yields significantly better results as compared to the non-hybrid GA.

As in the hybridization of the GA, the IGA was employed and significantly better results were obtained, the IGA was also independently run in order to realize whether the GA played any role in the overall performance of the proposed hybrid algorithm. If employing the GA has not been helpful at all, one would prefer to allocate all the mentioned CPU time to the IGA. This question will be addressed in the next sections. More specifically, the performance of the stand-alone IGA was tested on the benchmarks of Taillard and then its performance was compared with that of the hybrid GA.

\footnote{It is noteworthy that the IGA of Ruiz et al. [7] has been experimentally proved to be very effective. In fact, Ruiz et al. [7] compared the IGA against other twelve algorithms and they showed that their IGA outperformed all of them.}
Performance of the stand-alone IGA and its comparison with the proposed hybrid GA

A time-dependent termination condition of \( n \times m \times 90 \) milliseconds was considered for the implemented IGA, where \( n \) and \( m \) are the number of jobs and machines, respectively. In fact, this is the same termination condition adopted for the hybrid GA. The stand-alone IGA was run ten times for each of 110 instances of Taillard and the results, which are averaged on problem size, are given in Table 2. In this Table, the results of hybrid GA are also presented for the sake of comparison. As seen in Table 2, the average RPD obtained by the hybrid GA is at least as good as the average of RPD obtained by stand-alone IGA, except for the problem size of \( 200 \times 10 \).

The detailed comparative results for the instances are given in Tables 3-13. In these Tables, the results of 10 runs of the stand-alone IGA and hybrid GA are individually averaged for each instance.

From Table 3, it can be observed that both stand-alone IGA, and hybrid GA can obtain the optimal solution for all instances with the size \( 20 \times 5 \), except for instance labeled ta007. More specifically, the optimal solution for ta007 was not obtained after 10 trials (runs) of the hybrid GA. The stand-alone IGA could not achieve the optimal solution for ta007 either.

Table 4 shows that hybrid GA is slightly better than the stand-alone IGA on instances ta012, ta013, and ta020. Whereas, the stand-alone IGA outperforms the hybrid GA on ta018 with a slim margin.

It can be stated that the overall performance of the stand-alone IGA and hybrid GA are similar over the instances with the sizes of \( 20 \times 20 \), \( 50 \times 5 \), \( 50 \times 10 \), \( 50 \times 20 \), \( 100 \times 5 \), \( 100 \times 10 \). More specifically, in some cases, such as ta026 and ta032, the performance of the hybrid GA is slightly better than that of the stand-alone IGA. On the other hand, the performance of stand-alone IGA for some cases such as ta025 and ta041 was better than that of the hybrid GA. The slight advantage of the hybrid GA is more apparent for the instances with the size of \( 100 \times 20 \) and \( 200 \times 20 \). The corresponding results are given in Tables 11 and 13, respectively.

One should note that the advantage of the proposed hybrid GA is its robustness. This can be realized through the Figures 3-7. In Figures 3-5, it can be observed that different treatments of the factors do not change the obtained RPDs to a great extent. More specifically, the difference of the worst and best values for the RPDs in Figures 3-5 are not greater than 0.1. Whereas, the non-hybrid GA shows much higher variation in terms of the RPD. Figures 6 and 7 show that these variations could reach up to 4. Therefore, it can be concluded that the proposed hybrid GA is robust with respect to its parameters.

Conclusions and Future Work

The goal of this research was to survey one of the most well-known scheduling problems, the permutation flow-shop scheduling problem. Makespan, as the most common objective function (i.e., performance
Table 2: Average RPD for different instance sizes of Taillard’s benchmarks obtained by the stand-alone IGA, and the hybrid GA.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Instance Size</th>
<th>Obtained RPD by stand-alone IGA</th>
<th>Obtained RPD by hybrid GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta001-ta010</td>
<td>20 × 5</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>ta011-ta020</td>
<td>20 × 10</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>ta021-ta030</td>
<td>20 × 20</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>ta031-ta040</td>
<td>50 × 5</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>ta041-ta050</td>
<td>50 × 10</td>
<td>0.78</td>
<td>0.73</td>
</tr>
<tr>
<td>ta051-ta060</td>
<td>50 × 20</td>
<td>1.23</td>
<td>1.18</td>
</tr>
<tr>
<td>ta061-ta070</td>
<td>100 × 5</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>ta071-ta080</td>
<td>100 × 10</td>
<td>0.28</td>
<td>0.26</td>
</tr>
<tr>
<td>ta081-ta090</td>
<td>100 × 20</td>
<td>1.72</td>
<td>1.63</td>
</tr>
<tr>
<td>ta091-ta100</td>
<td>200 × 10</td>
<td>0.21</td>
<td>0.23</td>
</tr>
<tr>
<td>ta101-ta110</td>
<td>200 × 20</td>
<td>1.63</td>
<td>1.54</td>
</tr>
</tbody>
</table>

Table 3: Average RPD obtained by the stand-alone IGA, and the hybrid GA for different Taillard’s instances with the size of 20 × 5.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Instance Size</th>
<th>Obtained RPD by stand-alone IGA</th>
<th>Obtained RPD by hybrid GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta001</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta002</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta003</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta004</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta005</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta006</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta007</td>
<td>20 × 5</td>
<td>0.40</td>
<td>0.40</td>
</tr>
<tr>
<td>ta008</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta009</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta10</td>
<td>20 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 4: Average RPD obtained by the stand-alone IGA, and the hybrid GA for different Taillard’s instances with the size of 20 × 10.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Instance Size</th>
<th>Obtained RPD by stand-alone IGA</th>
<th>Obtained RPD by hybrid GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta011</td>
<td>20 × 10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta012</td>
<td>20 × 10</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>ta013</td>
<td>20 × 10</td>
<td>0.11</td>
<td>0.07</td>
</tr>
<tr>
<td>ta014</td>
<td>20 × 10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta015</td>
<td>20 × 10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta016</td>
<td>20 × 10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta017</td>
<td>20 × 10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta018</td>
<td>20 × 10</td>
<td>0.15</td>
<td>0.22</td>
</tr>
<tr>
<td>ta019</td>
<td>20 × 10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta020</td>
<td>20 × 10</td>
<td>0.13</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 5: Average RPD obtained by the stand-alone IGA, and the hybrid GA for different Taillard’s instances with the size of 20 × 20.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Instance Size</th>
<th>Obtained RPD by stand-alone IGA</th>
<th>Obtained RPD by hybrid GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta021</td>
<td>20 × 20</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>ta022</td>
<td>20 × 20</td>
<td>0.03</td>
<td>0.04</td>
</tr>
<tr>
<td>ta023</td>
<td>20 × 20</td>
<td>0.10</td>
<td>0.09</td>
</tr>
<tr>
<td>ta024</td>
<td>20 × 20</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta025</td>
<td>20 × 20</td>
<td>0.05</td>
<td>0.11</td>
</tr>
<tr>
<td>ta026</td>
<td>20 × 20</td>
<td>0.07</td>
<td>0.04</td>
</tr>
<tr>
<td>ta027</td>
<td>20 × 20</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>ta028</td>
<td>20 × 20</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>ta029</td>
<td>20 × 20</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>ta030</td>
<td>20 × 20</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 6: Average RPD obtained by the stand-alone IGA, and the hybrid GA for different Taillard’s instances with the size of 50 × 5.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Instance Size</th>
<th>Obtained RPD by stand-alone IGA</th>
<th>Obtained RPD by hybrid GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta031</td>
<td>50 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta032</td>
<td>50 × 5</td>
<td>0.07</td>
<td>0.05</td>
</tr>
<tr>
<td>ta033</td>
<td>50 × 5</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td>ta034</td>
<td>50 × 5</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>ta035</td>
<td>50 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta036</td>
<td>50 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta037</td>
<td>50 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta038</td>
<td>50 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ta039</td>
<td>50 × 5</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>ta040</td>
<td>50 × 5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 7: Average RPD obtained by the stand-alone IGA, and the hybrid GA for different Taillard’s instances with the size of 50 × 10.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Instance Size</th>
<th>Obtained RPD by stand-alone IGA</th>
<th>Obtained RPD by hybrid GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta041</td>
<td>50 × 10</td>
<td>1.23</td>
<td>1.32</td>
</tr>
<tr>
<td>ta042</td>
<td>50 × 10</td>
<td>1.54</td>
<td>1.33</td>
</tr>
<tr>
<td>ta043</td>
<td>50 × 10</td>
<td>1.11</td>
<td>1.05</td>
</tr>
<tr>
<td>ta044</td>
<td>50 × 10</td>
<td>0.16</td>
<td>0.15</td>
</tr>
<tr>
<td>ta045</td>
<td>50 × 10</td>
<td>0.88</td>
<td>0.92</td>
</tr>
<tr>
<td>ta046</td>
<td>50 × 10</td>
<td>0.42</td>
<td>0.30</td>
</tr>
<tr>
<td>ta047</td>
<td>50 × 10</td>
<td>0.89</td>
<td>0.84</td>
</tr>
<tr>
<td>ta048</td>
<td>50 × 10</td>
<td>0.28</td>
<td>0.28</td>
</tr>
<tr>
<td>ta049</td>
<td>50 × 10</td>
<td>0.28</td>
<td>0.23</td>
</tr>
<tr>
<td>ta050</td>
<td>50 × 10</td>
<td>1.03</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Table 8: Average RPD obtained by the stand-alone IGA, and the hybrid GA for different Taillard’s instances with the size of 50 × 20.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Instance Size</th>
<th>Obtained RPD by stand-alone IGA</th>
<th>Obtained RPD by hybrid GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ta051</td>
<td>50 × 20</td>
<td>1.30</td>
<td>1.24</td>
</tr>
<tr>
<td>ta052</td>
<td>50 × 20</td>
<td>1.14</td>
<td>1.07</td>
</tr>
<tr>
<td>ta053</td>
<td>50 × 20</td>
<td>1.56</td>
<td>1.36</td>
</tr>
<tr>
<td>ta054</td>
<td>50 × 20</td>
<td>1.38</td>
<td>1.03</td>
</tr>
<tr>
<td>ta055</td>
<td>50 × 20</td>
<td>1.19</td>
<td>1.23</td>
</tr>
<tr>
<td>ta056</td>
<td>50 × 20</td>
<td>1.30</td>
<td>1.28</td>
</tr>
<tr>
<td>ta057</td>
<td>50 × 20</td>
<td>1.19</td>
<td>1.11</td>
</tr>
<tr>
<td>ta058</td>
<td>50 × 20</td>
<td>1.47</td>
<td>1.37</td>
</tr>
<tr>
<td>ta059</td>
<td>50 × 20</td>
<td>1.11</td>
<td>1.11</td>
</tr>
<tr>
<td>ta060</td>
<td>50 × 20</td>
<td>0.65</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 9: Average RPD obtained by the stand-alone IGA, and the hybrid GA for different Taillard’s instances with the size of 100 × 5.
A genetic algorithm-based solution methodology was developed and implemented. More specifically, the performance of two versions of the proposed algorithm, namely stand-alone genetic algorithm (referred to as the non-hybrid genetic algorithm) and the hybrid genetic algorithm (i.e., hybridization of the GA with the iterated greedy search algorithm) were extensively studied. As for the comparative experimental results, Taillard’s standard benchmarks were employed. Experimental results presented in this work showed that the performance of the search for near optimal solutions was highly improved when the genetic algorithm adopted for solving the PFSP was hybridized with an iterated greedy search algorithm.

The parameters of the both hybrid and non-hybrid proposed genetic algorithms were individually tuned using the full factorial experimental design. As a result, it was shown that the following set of parameters yielded the optimal combination for the proposed hybrid GA: population size of 60, selection type of rank-based, crossover type of SBOX, crossover probability of 0.40, and mutation probability of 0.1, and IGA probability of 0.02.

As for the non-hybrid genetic algorithm, it was shown that the optimal combination of the parameters for the proposed non-hybrid GA was: population size of 60, selection type of rank-based, crossover type of SBOX, crossover probability of 0.40, and mutation probability of 0.20.

Furthermore, it was shown that the hybrid genetic algorithm performs very well on the benchmark problems of Taillard. The hybrid genetic algorithm obtains the optimal solutions for a large number of the small instances of Taillard (namely instances with the size of 20 × 5, 20 × 10, 20 × 20, 50 × 5, and 100 × 5). The hybrid genetic algorithm performs well for the larger instances as well. In the worst case, the relative percentage deviation from the best-known solution for the instances with the size of 200 × 20 is less than two. In addition, it was shown that the proposed hybrid genetic algorithm is robust with respect to its parameters.

The non-hybrid genetic algorithm is not as effective as the hybrid one in terms of the obtained relative percentage deviation from the best-known solution. As it was seen, the performance of the non-hybrid genetic algorithm was always worse than the hybrid genetic algorithm.

Finally, it was shown that the hybrid genetic algorithm performs slightly better than the iterative greedy search algorithm of Ruiz et al. [5]. It is noteworthy that Ruiz et al. [6] compared their iterative greedy search algorithm against other existing algorithms and concluded that their algorithm outperforms them.

Future work for this research could be the parallel implementation of the genetic algorithm and the iterative greedy search algorithm. In the parallel implementation, the genetic algorithm and the iterative greedy search algorithm can be run on two separate processing units where both algorithms have their own evolution but they exchange their best solution in different intervals during the time that algorithms are being run on two computers. Application of the proposed hybrid GA on the Traveling Salesman Problem is another application domain that has been drawing attention recently.

References


