A New Hyper-Heuristic to Generate an Effective Instance Genetic Algorithm for the Permutation Flowshop Problem

Abstract

In this paper, we propose HHGA, a new hyper-heuristic for the Permutation Flowshop Problem with makespan minimization. It consists of a high-level genetic algorithm which goal is to tailor a dedicated and effective genetic algorithm for each instance. Our experiments on well-known Taillard benchmarks showed the performance of tailored GAs when compared to state of the art algorithms.

1 Introduction

There has been a huge amount of work on heuristics and metaheuristics for a great number of combinatorial optimization problems, no method seems to perform well on all the studied problems. This was effectively proven by the No Free Lunch Theorem(Wolpert and Macready 1997). If a given heuristic shows good performance on a set of problems, it will have bad performance on others. As a result, a new methodology called Hyper-heuristics emerged. They operate at a higher level of abstraction(Wilson 2007). Instead of exploring the solutions search space, they explore the heuristics search space looking for the best performing heuristic for a problem or an instance of a problem.

A hyper-heuristic is an iterative process that chooses or generates the most appropriate heuristic that will be used to solve the current instance of the considered problem. Its architecture is composed of 2 major layers: (1) the Hyper-Heuristic Layer (HHL) selects or generates heuristics to solve the problem; (2) the Domain Problem Layer (DPL) which is composed on low level heuristics or heuristic components (Burke et al. 2013).

(Burke et al. 2013) considers two hyper-heuristics strategies: selection and generation. The former strategy selects, from a set of pre-defined heuristics, one or several ones to solve the problem while the second one builds a new heuristic based on the components of several heuristics.

The permutation flowshop scheduling problem (PFSP) is one of the most thoroughly studied scheduling problems in the OR literature. This layout implies a natural ordering of the machines in the shop in such a way that jobs go through the same machines in the same order (González-Neira, Montoya-Torres, and Barrera 2017). This problem was first studied by (Johnson 1954) where the specific case of two machines flowshop was optimally solved. However, the problem is known to be NP-hard for three or more machines. Since then, tremendous efforts had been devoted to design heuristic and metaheuristic methods to find high quality solutions within a reasonable computation time. The majority of researches on flowshop scheduling deals with makespan minimization and one can find in the literature several surveys presenting those works (Pan and Ruiz 2013; Gupta and Stafford 2006; Ruiz, Maroto, and Alcaraz 2005; Reza Hejazi and Saghafian 2005; Framinan, Gupta, and Leisten 2004). The most recent survey paper of (Fernandez-Viagas, Ruiz, and Framinan 2017) reports major results on this problem. The survey identifies and compares the best existing heuristics and metaheuristics.

However, it was pointed out that the use of hyper-heuristics for the resolution of both mono- and multi-machine scheduling problems has been scarce. We can cite that of (Salhi and Rodríguez 2014) which proposed a selection hyper-heuristic to specific instances of a scheduling problem using affinity and competence functions. For the hybrid flowshop problem in semiconductors back-end manufacturing, a genetic algorithm was adapted to the structure of the problem (Lin and Chen 2015) and a customized population-based metaheuristic was proposed to solve the lot-streaming flow shop scheduling problem (Meng et al. 2018). Genetic algorithms were also used to evolve dispatching rules to solve the Job shop scheduling problem in (Nguyen et al. 2013) and the dynamic multi-objective job shop scheduling problem in (Fang et al. 2014). However, few works on the use of hyper-heuristic for the resolution of permutation flowshop problem exist as the genetic programming based hyper-heuristic used to discover variants of the NEH procedure for flowshop (Vazquez-Rodriguez and Ochoa 2011) and the tabu search based hyper-heuristic of (Nugraheni and Abednego 2017).

Genetic algorithms (GA) are widely used in the OR literature. Many classical operators were proposed for optimization problems as in the work of (Goldberg and Deb 1991). In order to improve the performance of GAs, researchers have introduced new genetic operators that are specific to the problem they are solving and other works have hybridized their GA with a local search. Table 1 presents most referred GAs in the literature.

Table 1: Well knows GAs proposed in the literature

GA	Reference
Classical GAs	(Reeves 1995) (Iyer and Saxena 2004) (Chen, Vempati, and Aljaber 1995)
Specific GA	(Chen, Vempati, and Aljaber 1995) (Reeves and Yamada 1998) GA_RMA of (Ruiz, Maroto, and Alcaraz 2006)
GA hybridized with local search	(Ishibuchi et al. 1994) (Reeves and Yamada 1998) HGA_RMA of (Ruiz, Maroto, and Alcaraz 2006) CGA of (Nagano, Ruiz, and Lorena 2008) (Framinan and Pastor 2008) (Tseng and Lin 2010)

In each work, the proposed GA or even other metaheuristics were calibrated in order to find the most appropriate combination of parameters and operators that can give the best average quality of solutions for the studied problem.

Therefore, among the existing approaches in OR literature, we notice that, in general, some researchers proposed a standard genetic algorithm for an optimization problem and others introduced specific operators for the problem they were studying or they have hybridized their GA with another approach. on the basis of this, we ask the following question: is a standard GA sufficient for a given problem, for a benchmark, for a class of instances or for each instance?

We propose in this paper, HHGA, a new hyper-heuristic based on genetic algorithms to solve the PFSP aiming to find the most appropriate genetic algorithm for a given problem instance.

The rest of paper is organized as follows. Section 2 introduces the problem statement which is the permutation flowshop problem. In section 3, our approach is presented. Section 4 describes the experimentation made and a comparison with some of the best known and most recent heuristics and metaheuristic approaches for the PFSP reported in the literature highlights the performance of our approach. Finally, we summarize our results and present some ideas for a future work.

2 Problem Statement

In this paper, we address the scheduling problem in the permutation flowshop (PFSP) with makespan minimization. The PFSP can be viewed as a simplified version of the flowshop problem, and has been proved to be NP-hard (Garey, Johnson, and Sethi 1976). The problem can be formally defined as follows: A finite set $J = \{J_1, ..., J_n\}$ of n jobs must be processed on a finite set M of m machines $M = \{M_1, ..., M_m\}$ in a sequential manner starting from machine M_1 and finishing on machine M_m. The processing order of operations on machines is the same for all jobs i.e., all jobs are processed by all machines in the same order. Each job j, $j = 1..n$, requires a fixed and known, non-negative, amount of processing time on each machine i, $i = 1..m$, represented by p_{ij}. Furthermore, preemption is not allowed, each job is available and ready for processing at time zero and the setup times are sequence independent. In PFSP, the goal is to find a sequence i.e., a permutation of the numbers $1..n$ that optimizes completion times $Cmax$, also called makespan, of the last job; that is the total time to complete the schedule. When all jobs are scheduled, $C_{max} = C_{mn}$ the time at which machine M_m finishes processing the n^{th} job . This problem is denoted as $n|m|P|Fmax$ or as $Fm|prmu|C_{max}$ (Pinedo 2002), where m is the number of machines, n is the number of jobs, $prmu$ denotes that only permutation schedules are allowed, and C_{max}denotes the makespan minimization as the optimization criterion.

3 Proposed Approach

We propose HHGA, a new hyper-heuristic based on genetic algorithms to solve the PFSP which goal is to tailor a suitable genetic algorithm for a given problem instance. Its two layers are defined as follows:

1. **Hyper-heuristic layer** is a genetic algorithm evolving low-level genetic algorithms. It searches the space of possible genetic algorithms configurations.
2. **Domain problem layer** is composed of genetic operators that will be used to form a genetic algorithm dedicated to the PFSP instance.

In this section, we present first the strategy and design of the hyper-heuristic layer (HHGA) and then, the domain problem layer through their respective encoding scheme and genetic operators.

To distinguish between the two layers, we add the prefix hyper to the genetic operators and parameters of the hyper-heuristic layer.

3.1 Hyper-heuristic Layer

The effectiveness of a genetic algorithm greatly depends on the correct choice of the encoding, selection, crossover and mutation operators, as well as the probabilities by which they are applied. Calibrating genetic algorithms is necessary to get an algorithm which is more effective than all other configurations. In fact, if one takes the following possibilities for the different configurations of a low-level GA:

- Population size between 150 and 300,
- Numbers of generations between 70 and 150,
- 3 strategies for generating the initial population,
- 5 selection methods.
- 8 crossover methods.
- 6 mutation methods.
- Probability of crossover between 0.7 and 0.9.
- Probability of mutation between 0.01 and 0.15.

Then, without counting possibilities of crossover and mutation probabilities, all the cited factors result in a total of $151 \times 81 \times 3 \times 5 \times 8 \times 6 = 8806320$ different combinations and thus, 8806320 different genetic algorithms. To search for the most efficient GA in this search space, we designed, as a high-level strategy, a hyper-heuristic denoted HHGA. In other words, HHGA is used to tailor genetic algorithms for the PFSP's instances where individuals are low-level GAs that solve the PFSP.

The most remarkable characteristic of HHGA is that it obtains an effective genetic algorithm for an instance of PFSP by testing and evolving different possible combinations of genetic operators for this instance.

Algorithm 1: HHGA

```
Data:
hyper-Pop-Size : population size of HHGA
Hyper-N-gen : number of generations of HHGA
Hyper-Selection-method : hyper-selection method
Hyper-crossover-method : hyper-crossover method
Hyper-Replacement-method : hyper-replacement method
hyper-Pc: crossover probability
Hyper-Pm: mutation probability
Result: Tailored-GA for the problem instance
1  Generate initial population of Tailored-GA;
2  Evaluate population;
3  for $i = 1$ to Hyper-N-gen do
4      for $j = 1$ to hyper-Pop-Size do
5          $\alpha = get_R andom_N umber(0, 1)$;
6          if $\alpha \prec hyper - Pc$ then
7              p1, p2 = Hyper-Selection(hyper-Selection-method);
8              offSpring1, offSpring2=Hyper-Crossover (p1, p2, hyper-crossover-method);
9              Add offSpring1, offSpring2 to intermediate population.
10         end
11     end
12     for $k = 1$ to hyper-Pop-Size do
13         $\mu = get_R andom_N umber(0, 1)$;
14         if $\mu \prec hyper - Pm$ then
15             mutant = Hyper-mutation (hyper-population[k]);
16             add mutant to intermediate population;
17         end
18     end
19     Evaluate intermediate population;
20     population = Hyper-Replacement (hyper-population, intermediate-population, hyper-Replacement-method) ;
21 end
```

Encoding scheme and fitness function In HHGA, individuals are low level GAs coded as hyper-chromosomes. Each hyper-chromosome is presented as a vector of 10 genotypes (Table 1) and each genotype represents a genetic operator or a parameter as follows:

1. Population size (**PopSize**),
2. Number of generations (**NGen**),
3. Initialization method (**InitM**),
4. Selection method (**SelectionM**),
5. Crossover method (**CrossoverM**),
6. Probability of crossover (P_c),
7. Mutation method (**MutationM**),
8. Probability of mutation (P_m),
9. Local search method (**LSMethod**),
10. Probability of local search: Pl.

Figure 2 shows a vector of a hyper-chromosome with the value of each genotype. It presents a low-level GA having a population size of 150 individuals running over 70 generations (iterations). The initial generation is completely stochastic. The selection is performed using ranking method, the crossover applied is OX with a probability of 90%, the shift mutation is used with a probability of 5% and the insertion neighborhood is used with a probability of 50%.

HHGA needs a fitness function to evaluate the quality of a hyper-individual in the population. Since a hyper-individual presents a genetic algorithm, its fitness is defined as the fitness of its best solution obtained after NGen iterations.

HHGA Process HHGA is a top-level GA composed of a hyper-population of low-level GAs. The hyper-population evolves until a stopping criterion is reached. Various stopping conditions can be used in genetic algorithms such as maximum generations, elapsed time, stagnation (no change in fitness), etc. In our case, we use the maximum number of generations Hyper_NGen as a termination criterion.

The initial population of HHGA is composed by genetic algorithms generated randomly. The hyper-selection of parents is used to improve the quality of the hyper-population by selecting the fittest GAs. In this paper, we use the binary tournament selection (Goldberg and Deb 1991) that takes randomly two GAs and the best one is selected as parent. The hyper-crossover is performed between two GAs by using the standard crossover methods 1X, 2X and uniform crossover (Spears and De Jong 1995). For hyper-mutation, we defined a new mutation method adapted for hyper-chromosomes. The hyper-mutation selects randomly k genotypes in the hyper-chromosome and changes their values with random ones according to its domain of definition. For example, if this method selects the genotype that corresponds to crossover (crossover method) which has the value OX, then, the other available values (i.e. methods) that can be choosed are: 1X, 2X, PMX and LCS. An example of hyper-mutation is shown in Figure 3.

At the end of each generation, the population grows because each crossover of two hyper-chromosomes produces two new offsprings. Retaining only the best values from the resulting set tends to reduce the diversity and causes the algorithm to converge quickly to a local optimum. In order

Figure 1: Encoding scheme of a hyper-chromosome

PopSize	NGen	InitM	SelectionM	CrossoverM	Pc	MutationM	Pm	LSMethod	Pl

Figure 2: Example of a hyper-chromosome encoding

150	70	Stochastic	Ranking	OX	0.9	Shift	0.05	Insertion Neighborhood	0.5

Figure 3: Example of Hyper-mutation with k=3 where the algorithm chooses randomly the genes: population size, crossover method, and mutation method and changes the population size from 150 to 200, the crossover from OX to PMX and the shift mutation to position one

(a) A hyper-chromosome before hyper-mutation

150	70	stochastic	Ranking	LCS	0.8	Shift	0.05	Insertion Neighborhood	0.5

(b) mutated hyper-chromosome

200	70	stochastic	Ranking	PMX	0.8	Position	0.05	Insertion Neighborhood	0.5

to keep diversity, we apply the $\mu + \lambda$ replacement strategy which combines the previous parent population and the new generated chromosomes into the gene pool. Then, we select the best λ chromosomes from the combined population and consequently, the better low-level GAs are preserved to the next generation. The rest of the population is selected randomly. This allows a certain percentage of hyper-chromosomes with poor fitness values to be included in the population giving a chance to the GA to overcome local minima.

The outline of the proposed HHGA is given in Algorithm 1. The evaluation of the hyper-population (lines 10 and 23) is done in parallel to optimize the execution time.

Domain Problem Layer As described above, we use genetic algorithms as low-level heuristics. The domain problem layer is composed of GAs' operators of selection, crossover, mutation and replacement methods, and parameters as the population size, number of generations, and crossover and mutation probabilities.

GAs are effective despite the simplicity of their operators. Some blame them for having blind operators i.e. they do not have information of data on which they operate. To circumvent this, our approach integrates specific operators that were proposed for the PFSP and a local search method.

We present in Algorithm 2 the pseudo-algorithm of the low-level GA.

In the following subsections, we present the encoding scheme and fitness function. Then, we detail the specification of the different genetic operators we used for the low-level GAs' and the local search operator.

Encoding scheme and fitness function For the PFSP, we use a permutation π of jobs as a chromosome. For example, in a problem with six jobs and four machines, a permutation $\pi = [2, 3, 1, 6, 5, 4]$ is a chromosome that represents a scheduling in which the sequence of jobs on each machine is $J2, J3, J1, J6, J5, J4$. The fitness function is just the reciprocal of the objective function value, that is, the makespan or completion time as defined in Section 2.

Algorithm 2: Tailored-GA

```
Data: Pop-Size : population size
N-gen : number of generations
Selection-method : selection method
crossover-method : crossover method
$Pc, Pm$ : crossover and mutation probability
Result: Best Solution for the problem instance
1  Generate initial population of solutions
     (generation-method);
2  Evaluate population;
3  for $i = 1$ to N-gen do
4      for $j = 1$ to Pop-Size do
5          $\alpha = get_Random_Number(0, 1)$;
6          if $\alpha \prec Pc$ then
7              p1, p2 = Selection(Selection-method);
8              offSpring1, offSpring2=Crossover (p1,
                 p2, crossover-method);
9              Keep two best solutions in {p1,p2,
                 off-spring1, offspring2 } in actual
                 population ;
10         end
11     end
12     //mutate offsprings with a probability of $Pm$
       mutation (offspring1, MutationM, $Pm$)
       mutation (offspring2, MutationM, $Pm$)
       Insertion_Neighborhood_Search(offspring1,
       $Pls$)
       Insertion_Neighborhood_Search(offspring2,
       $Pls$) Replacement(ReplacementM) end
13 end
```

Generation of initial population Instead of starting with an initial population randomly generated, it seems more efficient to use special techniques to produce a higher quality in the initial population (Ruiz, Maroto, and Alcaraz 2006).In this paper, we use two strategies of generating the initial population defines as follows:

- Stochastic generation where all individuals are randomly generated,
- Semi-stochastic generation which consists on a three-step initialization procedure as follows: (1) we generate one chromosome (schedule) by NEH heuristic (Nawaz, Enscore, and Ham 1983); (2) we apply shift mutations (Taillard 1990) to the chromosome generated in the step 1 to generate 5% of the initial population. Shift mutation is the best mutation operator for PFSP among those evaluated by (Murata, Ishibuchi, and Tanaka 1996; Nearchou 2004); (3) Finally we generate the rest of the initial population $(PopSize \times (95\%))$ randomly.

Genetic process The population evolves over $NGen$ generations (iterations). At each iteration of the genetic algorithm, the following process is applied. First, parents are selected for crossover operator. We use three selection methods which are: Roulette, Ranking and K-tournament selection (Goldberg and Deb 1991). After that, the crossover generates new individuals by combining selected parents with a probability Pc, in order to generate better offspring chromosomes, i.e. better solutions. Many crossover operators have been proposed in the literature. In this study, we use classical methods PMX, OX, 1X and 2X crossovers (Davis 1985; Michalewicz and Hartley 1996) and other specific ones that are proposed for the flowshop SJOX, SBOX, SJ2OX and SB2OX (Ruiz, Maroto, and Alcaraz 2006). In the next step of the evolutionary process, a mutation is applied with a probability Pm. The mutation consists on a slight genotype change, it can also be seen as a simple form of local search. Mainly five different mutation operators were proposed in the literature for permutation encodings and that we use: shift, random exchange, position, displacement and inverse mutation (Nearchou 2004). Finally, the next population is formed where a weaker parent is replaced by a strong child. With the four individuals only the fittest two, parent or child, stay in the next population's generation.

Local search operator Intensification (exploitation) is one of the major issues introduced to build effective search algorithms. In order to intensify the search space, we apply a local search using the insertion neighborhood shown to be effective for the PFSP in (Taillard 1990). The insertion neighborhood (IN) goes through each job of the permutation and inserts it into all possible positions (Ruiz, Maroto, and Alcaraz 2006; Nowicki and Smutnicki 1996). In this approach, IN is applied on new solutions issued from the crossover and mutation with a probability Pl.

4 Computational Results

In this section, we report the result of a series of computational experiments, conducted to test the effectiveness of newly HHGA to tailor GAs for the PFSP instances. HHGA was coded in C# using Visual Studio 2015 Community 1 and the .NET framework 4.6. Experimentation was done on a computer with Windows 7 professional, 8 Go RAM and Intel Core i7-4790 processor running at 3.60 GHz.

The well-known flowshop benchmark of Taillard (Taillard 1993) was used to perform our experiments. Our test bed consists of a total of 10 subsets of this benchmark having the size $n \times m$ where $n \in 20, 50, 100, 200$ and $m \in \{5, 10, 20\}$. Therefore there are 100 instances, 10 of each particular configuration of n and m where the processing times (p_{ij}) are taken from a uniform distribution $U[1, 99]$ as it is common in flowshop scheduling research.

The performance of any solution is given by the relative percentage deviation (RPD) of the C_{max} provided by each GA with respect to the best-known solution for the instance to solve. RPD of an instance i is computed as follows:

$$RPDi = \frac{C_{max} - UpperBoundi}{UpperBoundi} \times 100 \qquad (1)$$

Where: C_{max} is the makespan of the best solution and $UpperBound_i$ is the makespan of the best-known solution for the instance i. All updates of the upperbound are available in Taillard's site[1]. To compare the quality of our approach with the literature, we use the average relative percentage deviation (ARPD) given by:

$$ARPDi = \frac{1}{N} \sum_{i=1}^{N} RPDi \qquad (2)$$

Where N is the number of problem instances

Table 2 summarizes the strategies used for each genotype of the hyper-heuristic.

4.1 Performance Analysis of HHGA

To test the effectiveness of tailored GAs generated by HHGA for solving each problem instance, we conducted intensive tests on the hyper-heuristic to analyze the tailored GAs that HHGA generates on a set of instances. Two main results have to be highlighted : (1) In almost all best generated tailored GAs, the initial population is generated with a semi-stochastic method and local search (LS) is always used. (2) There seems to be a link between an instance and its generated tailored GA as those GAs are different from one instance to another.

These observations led us to perform the following two tests: (1) we run HHGA on each instance of the 10 subsets of Taillard's Benchmark without using the local search on low-level GAs (with $Pl = 0$); then (2) we rerun HHGA on instances using the insertion neighborhood local search.

Table 3 shows the results obtained by HHGA with and without the local search. Since we are limited in number of pages, we did not include in this table, instances for which tailored GAs with and without LS reached an RPD of 0. Each instance is named as presented by Taillard (i.e. From tai001 to tai120).

Table 3: ARPD and ACPU obtained by each type of low-level GAs

	Instance	Tailored GA		Tailored GA with LS	
		RPD	CPU(s)	RPD	CPU(s)
20×5	tai007	0.40	0.33	0.40	3.49

[1]http://mistic.heig-vd.ch/taillard/

Table 2: Hyper-genes' strategies

Hyper-chromosomes gene	**Strategies**
Population size: **PopSize**	[50,400]
Number of generations: **NGen**	[200,600]
Initial generation	Stochastic, semi-stochastic
Selection method: **SelectionM**	Roulette, Ranking and K-tournament $K \in \{2, 3, 4\}$
Crossover method: **CrossoverM**	PMX, OX, 1X, 2X, SJOX, SBOX, SJ2OX, SB2OX, LCS
Probability of crossover: Pc	[0.6,0.9]
Mutation method: **MutationM**	shift, random exchange, position, displacement and inverse mutation
Probability of mutation: Pm	[0.01,0.2]
Local search method: **LSMethod**	Insertion neighborhood search
Probability of local search: Pl	[0,0.5]

	Instance	Tailored GA		Tailored GA with LS	
		RPD	CPU(s)	RPD	CPU(s)
50×5	tai032	0.14	0.77	0.07	8.66
	tai039	0.08	0.82	0.08	23.13
50×10	tai041	3.10	8.29	1.14	24.56
	tai042	2.70	2.82	1.53	40.04
	tai043	3.35	4.31	1.13	35.68
	tai044	0.52	5.33	0.13	32.13
	tai045	1.55	6.05	1.18	23.57
	tai046	1.93	49.28	0.00	36.27
	tai047	1.75	6.97	1.00	34.91
	tai048	0.66	6.50	0.16	24.74
	tai049	1.79	4.97	0.66	23.89
	tai050	2.15	11.94	0.85	30.59
50×20	tai051	3.09	5.06	1.25	1435.58
	tai052	3.80	4.44	0.89	1405.67
	tai053	4.59	4.98	1.73	32897.81
	tai054	3.38	5.14	1.40	4544.77
	tai055	4.54	6.30	1.44	798.98
	tai056	3.34	3.49	1.60	22317.63
	tai057	3.70	4.33	1.05	22782.55
	tai058	4.15	5.18	1.68	22080.39
	tai059	2.97	6.36	1.10	3534.74
	tai060	3.89	4.46	1.41	2314.71
100×5	tai063	0.08	12.99	0.00	68.77
	tai064	0.08	16.89	0.08	40.78
	tai067	0.02	18.71	0.00	106.55
100×10	tai071	0.26	5.53	0.00	7978.15
	tai072	0.37	6.65	0.02	6887.44
	tai073	0.30	12.91	0.05	5546.72
	tai074	1.04	3.41	0.26	20589.36
	tai075	1.13	3.93	0.04	46297.71
	tai076	0.26	4.83	0.09	34649.24
	tai077	0.71	3.50	0.02	22225.78
	tai078	1.14	8.75	0.23	14881.15
	tai079	0.55	3.21	0.09	85135.20
	tai080	0.62	6.03	0.00	17976.76
	tai081	4.14	16.26	2.35	35252.75
100×20	tai082	3.66	11.87	1.88	26134.86
	tai083	3.54	11.86	1.51	2938.77
	tai084	3.27	10.29	1.60	1749.64
	tai085	3.34	19.07	1.92	13015.55
	tai086	3.69	9.84	1.60	505.25
	tai087	3.59	7.21	1.69	3123.80
	tai088	4.25	12.21	2.28	1486.06
	tai089	3.84	9.34	2.10	4989.67
	tai090	2.72	17.92	1.46	8540.01
200×10	tai091	0.44	23.00	0.09	329.16
	tai092	0.98	77.12	0.28	483.10
	tai093	0.87	15.94	0.30	697.37
	tai094	0.41	68.61	0.04	693.69
	tai095	0.34	47.54	0.04	543.38
	tai096	0.50	55.09	0.05	334698.26
	tai097	0.35	45.76	0.06	857.17
	tai098	0.44	33.29	0.10	1155.81
	tai099	0.58	18.13	0.11	1440.90
	tai100	0.55	36.14	0.09	719331.18

From Table 3 , we can note that HHGA was able generate tailored GAs reaching the best known solutions for many problem instances without local search. In fact, 61 instances out of 100 have a non-zero RPD, which is a very interesting result. When we integrate local search, tailored GAs obtained very close deviations to 0 for instances tai007 of 20×5, tai032 and tai039 of 50×5, tai044, tai045 and tai048 of 50×10, tai063, tai064 and tai047 of 100×5, the 10 instances of each 100×10 and 200×10 sets. We further note that for the hardest subsets 50×20 and 100×20, these results are very promising to achieve better ones by adding some enhancements.

For each instance we compute the percentage of improvement as follows :

$$Improvement\_rate = \frac{RPD_{TGA} - RPD_{TGA\_LS}}{RPD_{TGA}} \times 100 \qquad (3)$$

where RPD_{TGA} is the RPD reached by tailored GA and RPD_{TGA_LS} is the RPD of the tailored GA using the local search.

Table 4: General configurations obtained by HHGA for the instances from tai010 to tai015

Instance	Ngen	PopSize	Initial method	Local search probability	Selection method	Crossover method	Pc	Mutation method	Pm
tai010	[400-500]	[100-200]	semi-stochastic	[0.17-0.5]	ranking	classical - dedicated	[0.5, 0.65]	position	[0.02-0.15]
tai011	[290-400]	[70-100]	semi-stochastic	[0.45-0.55]	roulette	dedicated	0.41	random exchange - position	0.1
tai012	[550-600]	[70-100]	semi-stochastic stochastic	[0.2 - 0.5]	three tournament -ranking	classical	[0.5 - 0.8]	position - shift	0.18
tai013	[500-550]	[50-70]	semi-stochastic	[0.3 - 0.5]	quadratic tournament - ranking	classical - dedicated	[0.6 - 0.7]	three inverse - random exchange	[0.06-0.15)
tai014	[450-500]	[70-80]	semi-stochastic	[0.3 - 0.5]	ranking	classical	[0.7 - 0.8]	position -random exchange	[0.09-0.15]
tai015	[200-350]	[100-150]	stochastic	[0.15 - 0.45]	two tournament	classical	[0.5 - 0.7]	three inverse - shift	[0.01-0.06)

Figure 4: Improvement rates for Taillard instances.

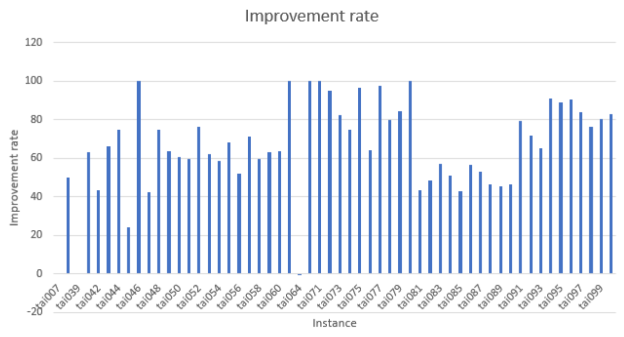

Figure 4 shows the improvement rates obtained by tailored GAs when using local search.

We notice that the local search brings a considerable improvement to the quality of tailored GAs even if it requires an important execution time. We can cite the example of instances tai046, tai063, tai067 and tai071 where an improvement of 100 % using local search reached an RPD of 0 for these instances.

Furthermore, the reader can observe that the rates of improvement are highly significant. Improvement rates are at least of zero (an improvement rate of zero means that we had the same quality with and without the local search) and in most instances, the local search had an improvement of more than 50%.

4.2 Analysis of obtained tailored GAs

Here, we zoom on some instances of the benchmark set 20×10 and we extract the general configuration of the tailored GAs that HHGA have generated. This configuration is retrieved from the best GAs of the last 5 generations of the hyper-heuristic.

Table 4 shows the extracted general configuration of the tailored GAs obtained for instances tai010 - tai015. we note that for numerical parameters like the population size ($PopSize$) we extract the range of values and for genetic operators we extract the operator that appears the most.

For example, for the instance tai010, the best tailored GAs have a population size in [100-200] performed over a number of generations between 400 and 500. The generation of the initial population is semi-stochastic, the selection method is the ranking one and the probabilities of crossover, mutation and local search are respectively in the ranges [0.5, 0.65] , [0.02-0.15] and [0.17-0.5].

In general, we can observe that each instance has specific configurations of tailored GAs. We can cite the example of the instance tai015; in this case, the combination of a stochastic initialization with a classical crossover, binary tournament selection and three inverse or shift mutation gave best results.

4.3 Comparative study

In this section, we compare tailored GAs of HHGA to the best genetic algorithm HGA_RMA of (Ruiz, Maroto, and Alcaraz 2006) and most frequently referred algorithms for the PFSP. The compared methods are: PACO of (Rajendran and Ziegler 2004), the iterated greedy IG_RS_{LS} of (Ruiz and Stützle 2007) and PSOvns of (Tasgetiren et al. 2007) . The results, averaged by subsets of Taillards benchmark for the compared methods are shown in Table 5. Results in bold refer to the best performing approaches among the compared ones.

Table 5: Comparison of different approaches with makespan objective

Benchmark	PACO	HGA RMA	IG_RS LS	PSO$_{vns}$	HHGA
20 × 5	0.18	0.04	0.04	**0.03**	0.04
20 × 10	0.24	0.02	0.06	0.02	**0.00**
20× 20	0.18	0.05	0.03	0.05	**0.00**
50 × 5	0.05	**0.00**	**0.00**	**0.00**	0.01
50 × 10	0.81	0.72	**0.56**	0.57	0.78
50 × 20	1.41	0.99	**0.94**	1.36	1.35
100 × 5	0.02	0.01	0.01	**0.00**	0.01
100 × 10	0.29	0.16	0.20	0.18	**0.08**
100 × 20	1.93	**1.30**	**1.30**	1.45	1.82
200×10	0.23	0.14	**0.12**	0.18	**0.12**
Average	0.53	0.34	**0.33**	0.38	0.42

We notice that our approach outperforms the others and reached the RPD of zero for the 10 instances of each 20×10 and 20×20 sets and is had best ARPD for 100×10 and 200×10 sets. Besides , it is as efficient as best approaches for the 20×5 and 100×5 sets. By analyzing the results above, the problem was in the 7th instance of 20×5 and instances tai063, tai064 and tai067 of 100×5 set. Furthermore, our approach is competitive for the other sets as 50×5 (instances tai032 and tai039) and 50×10 sets. However, HHGA as the other approaches does not have good results for the sets 50×20 and 100×20. This can be due to the difficulty of their problem instances.

5 Conclusion and Future Work

we proposed in this paper a new hyper-heuristic called HHGA to generate an effective genetic algorithm for each problem instance. We treated the permutation flowshop problem with makespan objective. On average, HHGA have reached 0.42% of taillard upper bounds and obtained and RPD of 0 for 39 instances. Through the experimentation, we can note that the local search and the semi-stochastic generation had a good influence on results obtained by tailored genetic algorithms. Moreover, we showed over tests the competitive results of HHGA with state of art approaches.

Nevertheless, we can further improve the current results by adding other mechanisms to either intensify or diversify the search space according to the variation of the actual generation of the low-level GAs in order to avoid their premature convergence and escape local optima. we can also analyze the hardest instances on which HHGA didn't reach good results by data mining approaches. Besides, we can test our approach on the newest benchmark named VFR of (Vallada, Ruiz, and Framinan 2015).

References

Burke, E. K.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; and Qu, R. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* 64(12):1695–1724.

Chen, C.-L.; Vempati, V. S.; and Aljaber, N. 1995. An application of genetic algorithms for flow shop problems. *European Journal of Operational Research* 80(2):389–396.

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

Fang, N.; Zhou, J.; Zhang, R.; Liu, Y.; and Zhang, Y. 2014. A hybrid of real coded genetic algorithm and artificial fish swarm algorithm for short-term optimal hydrothermal scheduling. *International Journal of Electrical Power & Energy Systems* 62:617–629.

Fernandez-Viagas, V.; Ruiz, R.; and Framinan, J. M. 2017. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research* 257(3):707–721.

Framinan, J. M., and Pastor, R. 2008. A proposal for a hybrid meta-strategy for combinatorial optimization problems. *Journal of Heuristics* 14(4):375–390.

Framinan, J. M.; Gupta, J. N.; and Leisten, R. 2004. A review and classification of heuristics for permutation flowshop scheduling with makespan objective. *Journal of the Operational Research Society* 55(12):1243–1255.

Garey, M. R.; Johnson, D. S.; and Sethi, R. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research* 1(2):117–129.

Goldberg, D. E., and Deb, K. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1. Elsevier. 69–93.

González-Neira, E.; Montoya-Torres, J.; and Barrera, D. 2017. Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations* 8(4):399–426.

Gupta, J. N., and Stafford, E. F. 2006. Flowshop scheduling research after five decades. *European Journal of Operational Research* 169(3):699–711.

Ishibuchi, H.; Yamamoto, N.; Murata, T.; and Tanaka, H. 1994. Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems. *Fuzzy Sets and systems* 67(1):81–100.

Iyer, S. K., and Saxena, B. 2004. Improved genetic algorithm for the permutation flowshop scheduling problem. *Computers & Operations Research* 31(4):593–606.

Johnson, S. M. 1954. Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics (NRL)* 1(1):61–68.

Lin, J. T., and Chen, C.-M. 2015. Simulation optimization approach for hybrid flow shop scheduling problem in semiconductor back-end manufacturing. *Simulation Modelling Practice and Theory* 51:100–114.

Meng, T.; Pan, Q.-K.; Li, J.-Q.; and Sang, H.-Y. 2018. An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem. *Swarm and Evolutionary Computation* 38:64–78.

Michalewicz, Z., and Hartley, S. J. 1996. Genetic algorithms+ data structures= evolution programs. *Mathematical Intelligencer* 18(3):71.

Murata, T.; Ishibuchi, H.; and Tanaka, H. 1996. Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering* 30(4):1061–1071.

Nagano, M. S.; Ruiz, R.; and Lorena, L. A. N. 2008. A constructive genetic algorithm for permutation flowshop scheduling. *Computers & Industrial Engineering* 55(1):195–207.

Nawaz, M.; Enscore, E. E.; and Ham, I. 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* 11(1):91–95.

Nearchou, A. C. 2004. The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics* 88(2):191–203.

Nguyen, S.; Zhang, M.; Johnston, M.; and Tan, K. C. 2013. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation* 17(5):621–639.

Nowicki, E., and Smutnicki, C. 1996. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research* 91(1):160–175.

Nugraheni, C. E., and Abednego, L. 2017. A tabu-search based constructive hyper-heuristics for scheduling problems in textile industry. *Journal of Industrial and Intelligent Information Vol* 5(2).

Pan, Q.-K., and Ruiz, R. 2013. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research* 40(1):117–128.

Pinedo, M. 2002. Scheduling: Theory, algorithms and systems. *2nd ed. Prentice-Hall, Englewood Cliffs, NJ*.

Rajendran, C., and Ziegler, H. 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research* 155(2):426–438.

Reeves, C. R., and Yamada, T. 1998. Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary computation* 6(1):45–60.

Reeves, C. R. 1995. A genetic algorithm for flowshop sequencing. *Computers & operations research* 22(1):5–13.

Reza Hejazi, S., and Saghafian, S. 2005. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research* 43(14):2895–2929.

Ruiz, R., and Stützle, T. 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research* 177(3):2033–2049.

Ruiz, R.; Maroto, C.; and Alcaraz, J. 2005. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research* 165(1):34–54.

Ruiz, R.; Maroto, C.; and Alcaraz, J. 2006. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega* 34(5):461–476.

Salhi, A., and Rodríguez, J. A. V. 2014. Tailoring hyper-heuristics to specific instances of a scheduling problem using affinity and competence functions. *Memetic Computing* 6(2):77–84.

Spears, W. M., and De Jong, K. D. 1995. On the virtues of parameterized uniform crossover. Technical report, NAVAL RESEARCH LAB WASHINGTON DC.

Taillard, E. 1990. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research* 47(1):65–74.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *european journal of operational research* 64(2):278–285.

Tasgetiren, M. F.; Liang, Y.-C.; Sevkli, M.; and Gencyilmaz, G. 2007. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European journal of operational research* 177(3):1930–1947.

Tseng, L.-Y., and Lin, Y.-T. 2010. A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. *International Journal of Production Economics* 127(1):121–128.

Vallada, E.; Ruiz, R.; and Framinan, J. M. 2015. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research* 240(3):666–677.

Vazquez-Rodriguez, J. A., and Ochoa, G. 2011. On the automatic discovery of variants of the neh procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society* 62(2):381–396.

Wilson, J. 2007. Search methodologies: introductory tutorials in optimization and decision support techniques.

Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* 1(1):67–82.