A new constructive heuristic for the No-Wait Flowshop Scheduling Problem

Lucien Mousin, Marie-Eléonore Kessaci, Clarisse Dhaenens

▶ **To cite this version:**

Lucien Mousin, Marie-Eléonore Kessaci, Clarisse Dhaenens. A new constructive heuristic for the No-Wait Flowshop Scheduling Problem. 11th Learning and Intelligent OptimizatioN Conference, Jun 2017, Nizhny Novgorod, Russia. hal-01579775

HAL Id: hal-01579775
https://hal.science/hal-01579775

Submitted on 31 Aug 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A new constructive heuristic for the No-Wait Flowshop Scheduling Problem

Lucien Mousin[1], Marie-Eléonore Kessaci[1], Clarisse Dhaenens[1]

Université de Lille, INRIA, CNRS, UMR 9189 - CRIStAL, France
{lucien.mousin}@ed.univ-lille1.fr
{me.kessaci, clarisse.dhaenens}@univ-lille1.fr

Abstract. Constructive heuristics have a great interest as they manage to find in a very short time, solutions of relatively good quality. Such solutions may be used as initial solutions for metaheuristics for example. In this work, we propose a new efficient constructive heuristic for the No-Wait Flowshop Scheduling Problem. This proposed heuristic is based on observations on the structure of best solutions of small instances as well as on analyzes of efficient constructive heuristics principles of the literature. Experiments have been conducted and results show the efficiency of the proposed heuristic compared to ones from the literature.

1 Introduction

Scheduling problems is an important class of combinatorial optimization problems, most of them being $NP-hard$. They consist in the allocation of different operations on a set of machines over the time. The aim of scheduling problems is to optimize different criteria such as the makespan or the flowtime. Among these scheduling problems, some of them, for example the jobshop or the flowshop, have been widely studied in the literature, as they represent many industrial situations. In this paper, we are specifically interested in the No-Wait Flowshop Scheduling Problem (NWFSP). This extension of the classical permutation Flowshop Scheduling Problem (FSP) imposes that operations have to be processed without any interruption between consecutive machines. This additional constraint aims at describing real process constraints that may be found in the chemical industry for example. Regarding the solving of the problem, this additional constraint also introduces interesting characteristics that we propose to exploit within the design of an efficient constructive heuristic.

Indeed, while solving $NP-hard$ problems, the use of exact methods, mostly based on enumerations, is not practicable. Therefore heuristics and metaheuristics are developed. Heuristics designed for a specific problem can use, for example, priority rules to construct the schedule. Their advantages are their speed – most of them are greedy heuristics – and their specificity – characteristics of the problem to be solved can be exploited –. Metaheuristics, on their side, are generic methods that can be applied to many optimization problems. Their efficiency is linked to their high ability of exploring the search space and the way

the metaheuristic has been adapted to the problem under study. The initial solution, when one is required, may also influence the quality of the final solution provided.

Hence, as explained, the quality of solutions provided by heuristics and metaheuristics depends on the way these methods can integrate specificities of the problem under study. Several approaches exist to integrate such specificities. For example, it is possible to analyze off line caracteristics of instances and to identify the best heuristic (or parameters of metaheuristic) for each kind of instances. Then, during the search, the method can identify the type of instance to solve and adapt the heuristic to use. This has been successfully used, for example, for a set partitioning problem [8]. Another approach deals with the integration of knowledge about interesting structures of solutions within the optimization method. In this context, the aim of the present work is to develop such a heuristic method for the No-Wait Flowshop Scheduling Problem. Thus, the contributions are the following: 1/ analysis of interesting structure of optimal solutions, 2/ integration of these observations in the design of a new constructive heuristic, 3/ study the interest of the proposed approach when used alone or as the initialization of a metaheuristic.

The remainder of this paper is organized as follows. Section 2 introduces the problem formulation and provides a literature review on constructive heuristics for the NWFSP. Section 3 presents in details the proposed constructive heuristic. In Section 4 experiments are conducted and a comparison with efficient heuristics of the literature is provided. Some conclusions and perspectives for future works are given in Section 5.

2 The No-Wait Flowshop Scheduling Problem

2.1 Description of the problem

The No-Wait Flowshop Scheduling Problem (NWFSP) is a variant of the well-known Permutation Flowshop Scheduling Problem (FSP), where no waiting time is allowed between the executions of a job on the successive machines. It may be defined as follows: Let J be a set of n jobs that have to be processed on a set of M ordered machines; $p_{i,j}$ is defined as the *processing time* of job i on machine j. A solution of the NWFSP is a sequence of jobs and so, is commonly represented by a permutation $\pi = \{\pi_1, \ldots, \pi_n\}$ where π_1 is the first job scheduled and π_n the last one. In this paper, the goal is to find a sequence that minimizes the makespan criterion (recorded as C_{max}) defined as the total completion time of the schedule.

Regarding the complexity of the problem, it has been proved by Wismer that the NWFSP can be viewed as an *Asymmetric Traveling Salesman Problem* (*ATSP*) [17]. In addition, Röck [13] proved that for m-machines ($m \geq 3$), the NWFSP is NP-hard while, the 2-machines case can be solved in $O(n * log\ n)$ [4].

However, the NWFSP possesses a characteristic not present in the classical FSP. Indeed, the start-up interval between two defined consecutive jobs of the

sequence on the first machine is constant and does not depend on their position within the sequence. This interval, called the delay between two jobs, has been defined by Bertolissi [2]. Let $d_{i,i'}$ be the delay of two jobs i and i', it is computed as follows:

$$d_{i,i'} = p_{i,1} + \max_{1 \leq r \leq m} \left(\sum_{j=2}^{r} p_{i,j} - \sum_{j=1}^{r-1} p_{i',j}, 0 \right)$$

This allows to compute the completion time $C_i(\pi)$ of the job π_i of sequence π directly from the delays of the preceding jobs as follows:

$$C_i(\pi) = \sum_{k=2}^{i} d_{\pi_{k-1},\pi_k} + \sum_{j=1}^{m} p_{\pi_i,j} \tag{1}$$

where $i \in \{2, ..., n\}$. Note that $C_1(\pi)$ is the sum of the processing times on the m machines of the first scheduled job and then, is not concerned by the delay. Therefore, the makespan ($C_{max}(\pi)$) of a sequence π can be computed from equation (1) with a complexity of $O(n)$.

Experiments we will present, involve local search methods at different steps. These methods need a neighborhood operator to move from a solution to another. Such neighborhood operators are specific to the problem and more precisely to the representation of a solution. In this work, we use a permutation representation and a neighborhood structure based on the *insertion operator* [14]. For a sequence π, it consists in inserting a job π_i at position k ($i \neq k$). Hence, jobs between positions i and k are shifted. The size of this neighborhood, *i.e.,* the number of neighboring solutions, is $(n-1)^2$. Two sequences π and π' are said to be neighbors when they differ from exactly one insertion move. It is very interesting to note that exploiting the characteristics of the NWFSP, the makespan of π' can be directly computed from the makespan of π with a complexity of $O(1)$ [11].

2.2 State-of-the-art

Many heuristics and metaheuristics have been proposed to solve scheduling problems and in particular the No-Wait Flowshop variant. The literature review proposed here, focuses on constructive heuristics as it is the scope of the article.

First, let us note that the well-known NEH (Nawaz, Enscore, Ham) heuristic, proposed for the classical permutation FSP [10], has been successfully applied on the No-Wait variant. Moreover, constructive heuristics have been designed specifically for the NWFSP. We may cite BIH (Bianco et al. [3]), BH (Bertolissi [2]) and RAJ (Rajendran [12]) among others. These heuristics define the order in which jobs are considered, regarding one criterion (e.g. decreasing order of sum of processing time of each job i - $\sum_m p_{i,m}$), but this may be improved with other jobs orders, even random. Indeed, in some contexts, it has been shown that repeating a random construction of solutions, may be more efficient than applying a constructive heuristic [1].

All these heuristics construct good quality solutions that can be further improved by a neighborhood-based metaheuristic, for example. In this paper, we focus on NEH and BIH as they provide high quality solutions and are considered as references for this problem. These two heuristics will be used later in the article for comparison.

The principle of NEH heuristic is to iteratively build a sequence π of jobs J. First, the n jobs of J are sorted by decreasing sums of processing times. Then, at each iteration, the first remaining unscheduled job is inserted in the partial sequence π in order to minimize the partial makespan. Algorithm 1 presents NEH.

Algorithm 1: NEH

Data: Set J of n jobs, π the sequence of jobs scheduled
$\pi = \emptyset$;
Sort the set J in decreasing sums of processing times;
for $k = 1$ ***to*** n **do**
  Insert job J_k in π at the position, which minimizes the partial makespan.

This is a greedy heuristic; only n iterations are needed to build a sequence. At each iteration, a job J_k is inserted. The makespan of the new partial sequence, may be computed with a complexity of $O(n)$ thanks to the delay, when job J_k is inserted at the first position (see section 2.1). Partial sequences, in which J_k is inserted at another position in the sequence, are neighbors of the previous one so, their makespan may be computed with a complexity of $O(1)$, according to the property exposed at the end of Section 2.1. Thus the complexity of one iteration of NEH is $O(n)$. Therefore, the complexity of the whole execution of NEH is $O(n^2)$.

BIH heuristic is another constructive heuristic based on the best insertion of a job in the partial sequence. The main difference with NEH is that all the unscheduled jobs are tested to find the best partial makespan. Indeed, while unscheduled jobs are remaining, for each one, all the possible positions to insert it into the partial sequence are evaluated ; the job and the position that minimize the partial makespan are chosen. Algorithm 2 presents BIH.

If we consider the schedule of a job as one iteration, BIH builds a sequence with n iterations only. However, at each iteration, each job is a candidate to be inserted at one of all the possible positions to find the one that minimizes the partial makespan. For each job, finding the position that minimizes the partial makespan has a maximal complexity of $O(n)$ so, the complexity of one iteration of BIH is $O(n^2)$. Therefore, the complexity of the whole execution of BIH is $O(n^3)$.

Algorithm 2: BIH

Data: Set J of n jobs, π the sequence of jobs scheduled
$\pi = \emptyset$;
while $J \neq \emptyset$ **do**
  Find job $k \in J$, which can be inserted in the sequence π, such that the partial makespan is minimized. Let h be the best insertion position of job k in the sequence π;
  Insert job k at position h in the sequence π;
  Remove k from set J;

BIH has been proposed in the literature after NEH in order to construct better quality solutions. However, the complexity of BIH is $O(n^3)$ whereas the one of NEH is $O(n^2)$. Thus, NEH is much faster than BIH, and for large size problems a compromise between computational time and quality is required.

3 IBI: Iterated Best Insertion heuristic

Following the construction's principles of previously exposed heuristics and after analyzing the structure of optimal solutions, we propose the Iterated Best Insertion (IBI) heuristic.

3.1 Analysis of optimal solutions structure

Most of constructive heuristics are based on the best insertion principle. Indeed, the principle of the heuristic is to increase to increase, at each iteration, the problem size by one. It starts with a sub-problem of size one (only one job to schedule), then increases the size of the problem to solve (two jobs to schedule, three jobs...) until the size of the initial problem is reached.

In order to understand the dynamic of such a construction, we analyzed the structure of consecutive constructed sub-sequences and compare them to optimal sequences. Obviously this can be done only for first steps, as it is impossible to enumerate all the sequences, and find the optimal one, when the problem size is too large. Our proposition is to extend observations realized on sub-problems to the whole problem, in order to provide better solution. This approach can be compared to the Streamlined Constraint Reasoning, where the solution space is partitioned into sub-spaces whose structures are analyzed to better solve the whole problem [5].

Figure 1 gives an example starting from a sub-problem of size 8, $\mathcal{P}_8$, where 8 jobs are scheduled in the optimal order to minimize the makespan. Then, following the constructive principle, job 9 is inserted at the position that minimizes the makespan leading to the sub-problem $\mathcal{P}_9$ of size 9. This sequence corresponds to the one given by NEH strategy. When comparing the obtained sequence with the optimal solution of $\mathcal{P}_9$, we can observe that they are very close. Indeed, only

Fig. 1. Example of the evolution of the structure of the optimal solution from a sub-problem $\mathcal{P}_8$ of size 8 to a sub-problem $\mathcal{P}_9$ of size 9.

$\mathcal{P}_8$		2	7	8	4	3	1	5	6		$C^*_{max} = 350$	optimal solution
	insertion of job 9	2	7	**9**	8	4	3	1	5	6	$C_{max} = 447$	
$\mathcal{P}_9$	re-insertion of job 7	2		9	8	4	3	1	5	6 **7**	$C_{max} = 435$	
	re-insertion of job 2			9	8	4	3	1	5	6 7 **2**	$C^*_{max} = 427$	optimal solution

two improving re-insertions (re-insertions of jobs 7 and 2) are needed to obtain this optimal solution from the NEH solution.

Experimental analysis on different instances have confirmed the following observations:

- at each step the structure of the obtained solution is not far from the structure of the optimal solution for the sub-problem,
- a few improving neighborhood applications may often lead to this optimal solution.

3.2 Design of IBI

Following these observations, we propose a new constructive heuristic called Iterated Best Insertion (IBI). At each iteration, two phases are successively achieved: (i) the first remaining unscheduled job is inserted in the partial sequence in order to minimize the partial makespan (same strategy than NEH) and (ii), an iterative improvement is performed on this partial solution to re-order jobs and to expect to be closer to the optimal solution of the sub-problem (see Algorithm 3). An iterative improvement is a method that moves, using a neighborhood operator, from a solution to one of its improving neighbors, which optimizes the quality. It naturally stops when a local optimum (a solution with no improving neighbors) is reached. Here, the neighborhood operator is based on the insertion and then possible move are applied while a better makespan is found.

Algorithm 3: IBI

```
Data: Set J of n jobs, π the sequence of jobs scheduled, σ criterion to sort J,
      cycle number of iterations without iterative improvement.
π = ∅;
Sort set J according to σ ;
for k = 1 to n do
    Insert job J_k in π at the position which minimizes the partial makespan.;
    if k ≡ 0 [cycle] then
        Perform an iterative improvement from π.
```

Two parameters have been introduced in the proposed method. First, σ a criterion to initially sort the set of jobs J. It will indicate in which order jobs

will be considered during the construction. Second, *cycle* the number of iterations without applying the iterative improvement procedure. Indeed, even if the experimental analysis shows that the sequence built in phase (i) is not far from the optimum of the sub-problem, it is known that the exploration of the neighborhood is more and more time-consuming and the optimum more and more difficult to reach with the increase of the problem size. In order to control the time-performance of IBI, the *cycle* parameter allows the iterative improvement to be executed at a regular number of iterations only.

3.3 Experimental Analysis of parameters

As the initial sort of IBI σ, may impact its performance as well as the *cycle* parameter, the performance of IBI is evaluated under these two parameters. This part, first presents the experimental protocol used, and then compare several versions of the IBI algorithm, using several initial sorts and finally analyses the impact of the cycle parameter.

Experimental Protocol The benchmark used to evaluate the performance of the different variants of IBI is the Taillard's instances [15], initially provided for the flowshop problem but also widely used in the literature for the NWFSP. This benchmark proposes 120 instances, organized by 10 instances of 12 different sizes. Our experiments are conducted on the 10 available instances of size `N_M` where the number of jobs `N` is ranging from 20 to 200 and the number of machines `M` from 5 to 20 (total of 110 instances).

To compare the algorithms, the Relative Percentage Deviation (RPD) with the best known solution is computed. For a quality Q to minimize, the RPD is computed as follows:

$$RPD = \frac{\text{Q(Solution)} - \text{Q(Best known solution)}}{\text{Q(Best known solution)}} * 100 \tag{2}$$

Because of its Iterative Improvement phase, IBI is a stochastic method. Thus, 30 runs are required to allow making conclusions about results obtained. To follow this recommendation, all variants of the algorithm are executed 30 times per instance and the statistical Friedman test is performed on the RPD of all executions to compare algorithms.

Each algorithm is implemented using C++ and the experiments were executed on an Intel(R) Xeon(R) 3.5GHz processor.

IBI: Initial Sort Greedy heuristics, such as IBI, consider jobs in a given order. For example, in NEH, jobs are ordered by the decreasing sum of processing times. The specificity of the NWFSP gives other useful information as the GAP between two jobs on each machine that represents the idle time of the machine between the two jobs [9]. This measure does not depend on the schedule. Hence, for each machine, the GAP between each pair of jobs can be computed independently

from the solving. The sum of GAPs for a pair of jobs represents the sum of their GAP on all the machines. To obtain a single value per job, we propose to define the *total GAP* of one job as the sum of the sum of GAPs. Hence, a high value of total GAP for a job indicates that the job has no good matching in the schedule. On the other hand, a low total GAP indicates that the job fits well with the others. In order to evaluate the impact of the initial sort and to define the most efficient one for the IBI heuristic, we experiment several initial sorts σ, based on (i) the total GAP or on (ii) the sum of processing times in decreasing or increasing order for both.

Table 1. Average RPD obtained on each size of Taillard's instances for different sorts: No Sort, Decreasing Mean GAP (DecrMeanGAP), Increasing Mean GAP (IncrMeanGAP), Decreasing sum of processing times (DecrPI) and the increasing sum of processing times (IncrPI). RPD values in bold stand for algorithms outperforming the other ones according to the statistical Friedman test.

Instance	NoSort	DecrMeanGAP	IncrMeanGAP	DecrPI	IncrPI
20_5	**1.79**	2.32	2.57	2.13	**1.38**
20_10	2.04	1.72	1.67	1.30	1.77
20_20	1.44	1.26	1.04	1.12	1.15
50_5	3.74	3.65	4.01	**3.23**	**3.04**
50_10	2.78	3.17	3.25	2.74	2.57
50_20	2.53	2.41	2.47	2.80	2.50
100_5	4.68	4.52	4.67	**4.13**	**4.12**
100_10	3.45	3.36	3.53	3.24	3.27
100_20	3.05	2.82	2.81	2.79	2.89
200_10	4.32	4.19	4.22	**3.81**	**3.87**
200_20	3.24	3.26	3.23	**3.03**	**2.99**

Table 1 provides a detailed comparison of the different initial sorts on the 110 Taillard's instances. This table shows that when a significant difference is observed the best initial sort is to consider the sum of processing times. Ordering jobs by increasing or decreasing order has no significant influence. Thus for the following, we choose the increasing sum of processing times as the initial sort of IBI.

IBI: Cycle The use of an iterative improvement at each iteration can be time expensive. In order to minimize the execution time, we introduce a cycle. A cycle is a sequence of x iterations without any iterative improvement.

In Table 2, we study the impact of the size of the cycle on both the quality of solutions and the execution time.

These results show that the quality decreases with a large cycle size, but is obviously faster. As the objective of such a constructive heuristic is to provide

Table 2. Average RPD and time (milliseconds) obtained on Taillard's instances for different sizes of cycle. RPD values in bold stand for algorithms outperforming the other ones according to the Friedman test. A cycle of size x, indicates the iterative improvement is executed every x iterations. Thus, a cycle of size n indicates the iterative improvement is executed only once, at the end of the construction.

	1		2		5		10		n	
Instance	RPD	time	RPD	time	RPD	time	RPD	time	RPD	time
20_5	1.38	0.77	1.24	0.55	1.43	0.27	1.78	0.20	1.62	0.18
20_10	1.77	0.83	1.82	0.52	1.85	0.26	1.49	0.20	1.69	0.19
20_20	1.15	0.77	1.35	0.47	1.71	0.25	1.74	0.19	1.68	0.16
50_5	**3.04**	12.16	**3.25**	6.76	3.46	3.52	3.62	2.49	3.94	1.74
50_10	2.57	11.55	2.49	6.69	2.75	3.62	3.07	2.39	3.26	1.48
50_20	2.50	11.73	2.37	6.67	2.58	3.58	2.65	2.42	3.14	1.59
100_5	**4.12**	95.74	4.22	55.96	4.42	30.22	4.65	19.40	5.29	9.62
100_10	**3.27**	93.43	**3.27**	54.56	3.30	29.19	3.45	18.78	4.17	9.71
100_20	**2.89**	92.90	**2.93**	53.60	**3.07**	28.35	3.16	18.28	3.56	9.88
200_10	**3.87**	755.09	**3.91**	435.40	4.07	225.52	4.09	146.26	4.74	55.04
200_20	**2.99**	746.62	3.10	431.33	3.11	222.80	3.18	146.26	3.93	57.83

in a very reasonable time a solution as good as possible, and as the time required here even for large instances stays reasonable, we propose not to use any cycle, that is to say to execute the iterative improvement at each step of the construction.

In conclusion of these experiments we propose to fix for the remainder of this work as IBI parameters: an initial sort based on the increasing sum of processing times and no cycle (*i.e.* cycle of size 1).

4 Experiments

The aim of this section is to analyze the efficiency of the proposed IBI method in two situations. First, IBI alone will be compared to other constructive heuristics of the literature, and in a second time these different heuristics will be used as initialization of a classical local search. Along this section, the same experimental protocol as before is used, and parameters used for IBI are those resulting from experiments of section 3.3.

4.1 Efficiency of IBI

We choose to compare IBI with two other heuristics of the literature: NEH and BIH. Indeed, as exposed in section 2.2 NEH and BIH are both interesting and efficient constructive heuristics for the NWFSP:

1/ NEH is a classical heuristic for flow-shop problems and it has the advantage to be very fast. Moreover, the proposed method IBI can be viewed as an improvement of NEH, as an iterative improvement is added at each step.

2/ BIH is a classical heuristic for the NWFSP and very efficient as, according to some preliminary experiments, it offers the best performance among several tested heuristics.

Table 3 shows the comparative study between IBI and the two other constructive heuristics. Both RPD and computation time are given to exhibit the gain in quality with regards to the time. NEH and BIH are deterministic and so, both values of an instance size are average ones computed from the ten RPD values obtained for the 10 instances respectively. On the other hand, IBI values correspond to the average ones over the 30 runs and the 10 instances of each instance size.

Table 3. RPD and time (milliseconds) obtained on Taillard's instances for NEH, BIH and IBI (average values). RPD values in bold stand for algorithms statistically outperforming the other ones according to the Friedman test.

	NEH		BIH		IBI	
Instance	RPD	time	RPD	time	RPD	time
20_5	3.95	0.03	3.03	0.10	**1.38**	0.77
20_10	4.18	0.02	3.60	0.09	**1.77**	0.82
20_20	2.92	0.02	**1.74**	0.08	**1.15**	0.76
50_5	6.84	0.10	5.98	1.10	**3.04**	12.16
50_10	4.59	0.09	4.10	1.13	**2.57**	11.54
50_20	4.84	0.10	4.01	1.18	**2.50**	11.72
100_5	8.14	0.32	6.85	9.07	**4.12**	95.74
100_10	6.10	0.33	5.66	9.29	**3.27**	93.43
100_20	5.17	0.33	4.65	9.30	**2.89**	92.90
200_10	6.72	1.21	5.85	72.44	**3.87**	755.09
200_20	5.74	1.21	4.51	71.95	**2.99**	746.62

Solutions built by IBI have a better quality (lower RPD) than those built by NEH or BIH. It has statistically been verified with the Friedman Test. Beside, IBI was able to better perform than NEH or BIH for 106 instances over the 110 available instances. The counterpart of this performance is the computational time required to execute IBI regarding to the two other heuristics. However, this time remains reasonable as, less than one second is required even for larger instances. IBI is an efficient alternative to the classical heuristics used to build good quality solutions.

4.2 IBI as initialization of a local search

A general use of constructive heuristics is to execute them as an initialization phase of meta-heuristics. In order to evaluate the pertinence of using IBI in such an initialization phase, it has been combined with a Tabu Search (TS).

This local search is widely used to solve flowshop problems [16], [7], [6] and so is a good candidate to show the pertinence to build solutions with IBI rather than NEH or BIH. The three heuristics are combined with TS. In order to be fair, every combined approaches have the same running time fixed to $1000 * n$ milliseconds (with n the number of jobs). These experiments aim at checking if the quality reached is enough to justify the use of IBI as initialization or if its *higher* execution time penalizes its use.

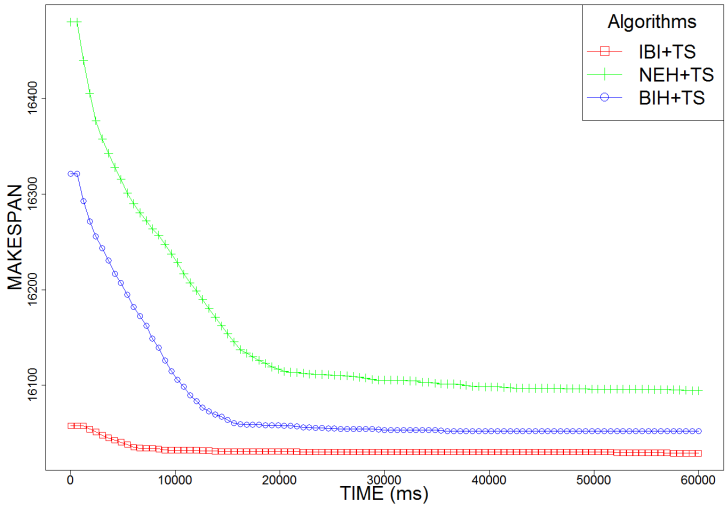
Table 4 presents results about the average RPD values among the 30 runs for each combined approach for each instance size. Results exposed in this table compared to those of Table 3 first indicate that the Tabu Search manages to improve solutions produced by the different heuristics. It also indicates that IBI used as an initialization phase, gets better results than the others, even if the difference is not always statistically proved. In particular for instances with a high number of jobs to schedule, IBI shows a very good performance.

Table 4. Average RPD on Taillard's instances for NEH+TS, BIH+TS and IBI+TS. RPD values in bold stand for algorithms statistically outperforming the other ones according to the Friedman test.

Instance	NEH+TS	BIH+TS	IBI+TS
20_5	1.12	1.30	0.83
20_10	1.30	1.35	1.39
20_20	1.15	0.71	0.73
50_5	3.17	3.49	2.63
50_10	2.68	2.53	2.15
50_20	2.53	2.41	2.03
100_5	4.66	4.55	**3.78**
100_10	3.70	3.48	**2.94**
100_20	2.97	2.93	2.62
200_10	4.25	4.20	**3.67**
200_20	3.56	**2.99**	**2.77**

As an illustration, Figure 2 shows the evolution of the average makespan for a particular instance with 200 jobs and 20 machines (mean of the 30 runs). Regarding only the quality after the initialization, IBI is better than the two other heuristics, as shown in Table 3 previously. After the improvement of the Tabu Search for the two other heuristics, the final quality seems to be equivalent to the one obtained by IBI without applying the local search. That shows the efficiency of this approach. Finally, the Tabu Search does not help a lot IBI. Indeed, thanks to the successive local improvement in the second phase of IBI, the solution provided by IBI is already a local optimum. Therefore, it is difficult for the Tabu Search to improve the solution obtained with IBI.

Fig. 2. Evolution of the average makespan on 30 runs for the 7th instance of Taillard (200 jobs, 20 machines) for the three combined approaches. IBI+TS is represented by squares, NEH+TS by crosses and BIH+TS by circles.



In conclusion, these experiments show the good performance of IBI for both aspects: as a constructive heuristics and as initialization of a meta-heuristic. In particular, they show the contribution of the local improvement to build the solution.

5 Conclusion and Perspectives

This work presents IBI, a new heuristic to minimize the makespan for the No-Wait Flowshop Scheduling Problem. IBI has been designed from the analysis of existing heuristics of the literature as well as of the structure of optimal solutions. Indeed, IBI is an improvement of the widely used heuristic (NEH), where an iterative improvement procedure has been added after each insertion of a job. Even if this additional procedure increases the computational time compared to other constructive heuristics of the literature (NEH and BIH), the improvement of the quality of the final solution built is noticeable and has statistically been validated with the Friedman test.

Then, we analyzed if this extra computational time is justified in term of quality obtained by evaluating the capacity of IBI to provide a good solution. IBI and the others heuristics have been used as initialization for a metaheuristic *i.e.*, the initial solution of the approach is a solution built by a heuristic. The experiments, on the Taillard instances, show that IBI helps the metaheuristic to be more efficient and the extra time needed to build an initial solution is not a drawback.

What is interesting in this work is that the addition of a simple modification of an existing constructive heuristic improves the results. This modification has been proposed from properties observed which make the application of an iterative improvement procedure at each step 1/ useful – only a few steps are required to reach the optimal solutions for the small sub-problems – and 2/ no time consuming – the makespan of a neighborhood solution can be computed in $O(1)$–. We can now wonder whether such a modification can be performed to other constructive heuristics for other variants of the flowshop or more generally, for other problems (if they also present equivalent properties).

References

1. Egon Balas and Maria C. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6):875–890, dec 1996.
2. Edy Bertolissi. Heuristic algorithm for scheduling in the no-wait flow-shop. *Journal of Materials Processing Technology*, 107(1-3):459–465, nov 2000.
3. Lucio Bianco, Paolo Dell'Olmo, and Stefano Giordani. Flow shop no-wait scheduling with sequence dependent setup times and release dates. *INFOR: Information Systems and Operational Research*, 37(1):3–19, feb 1999.
4. P. C. Gilmore and R. E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679, oct 1964.
5. Carla Gomes and Meinolf Sellmann. Streamlined constraint reasoning. In *Principles and Practice of Constraint Programming – CP 2004*, pages 274–289. Springer Nature, 2004.
6. Józef Grabowski and Jarosław Pempera. The permutation flow shop problem with blocking. a tabu search approach. *Omega*, 35(3):302–311, jun 2007.
7. Józef Grabowski and Mieczyslaw Wodecki. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, 31(11):1891–1909, sep 2004.
8. Serdar Kadioglu, Yuri Malitsky, and Meinolf Sellmann. Non-model-based search guidance for set partitioning problems. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012.
9. Marcelo Seido Nagano and Daniella Castro Araújo. New heuristics for the no-wait flowshop with sequence-dependent setup times problem. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 36(1):139–151, aug 2013.
10. Muhammad Nawaz, E Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, jan 1983.
11. Quan-Ke Pan, Ling Wang, M. Fatih Tasgetiren, and Bao-Hua Zhao. A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *The International Journal of Advanced Manufacturing Technology*, 38(3-4):337–347, jul 2007.
12. Chandrasekharan Rajendran. A no-wait flowshop scheduling heuristic to minimize makespan. *Journal of the Operational Research Society*, 45(4):472–478, apr 1994.
13. Hans Röck. The three-machine no-wait flow shop is NP-complete. *Journal of the ACM*, 31(2):336–345, mar 1984.

14. T. Schiavinotto and T. Stützle. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34:3143–3153, 2007.
15. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, jan 1993.
16. Chuyang Wang, Xiaoping Li, and Qian Wang. Accelerated tabu search for no-wait flowshop scheduling problem with maximum lateness criterion. *European Journal of Operational Research*, 206(1):64–72, oct 2010.
17. D. A. Wismer. Solution of the flowshop-scheduling problem with no intermediate queues. *Operations Research*, 20(3):689–697, jun 1972.