An Iterated Greedy-based Approach Exploiting Promising Sub-Sequences of Jobs to solve the No-Wait Flowshop Scheduling Problem

Lucien Mousin, Marie-Eléonore Kessaci, Clarisse Dhaenens

An Iterated Greedy-based Approach Exploiting Promising Sub-Sequences of Jobs to solve the No-Wait Flowshop Scheduling Problem

Lucien Mousin, Marie-Eléonore Kessaci, Clarisse Dhaenens

Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France
lucien.mousin@ed.univ-lille1.fr
{me.kessaci, clarisse.dhaenens}@univ-lille1.fr

Abstract

The no-wait flowshop scheduling problem is a variant of the classical permutation flowshop problem, with the additional constraint that jobs have to be processed by the successive machines without waiting time. To efficiently address this $NP - hard$ combinatorial problem we conducted an analysis of the structure of good quality solutions. This study shows that the No-Wait specificity gives them a common structure: they share identical sub-sequences of jobs. After a discussion on the way to identify these sub-sequences, we propose to exploit them into the well-known Iterated Greedy algorithm. Experiments are conducted on Taillard's instances. The experimental results show the proposed approach is efficient and robust, and is able to find out new best solutions for all the largest instances.

1 Introduction

Scheduling problems are classical combinatorial optimization problems. Most of them are NP-hard, and many advanced optimization methods (exact, heuristics and meta-heuristics) are proposed to deal with them. Within these approaches, some of them integrate some learning phases to guide the search to good quality regions of the search space.

The approach we propose here can enter within this category. Indeed, in this paper we are interested in solving the No-Wait Flowshop Scheduling Problem (NWFSP) which is a variant of the well-known Permutation Flowshop Scheduling Problem (PFSP). An analysis of efficient solutions of some instances (global and local optima) allows us to make an observation: sub-sequences of jobs are common within these solutions. This leads us to propose the concept of super-jobs (sub-sequences of consecutive jobs) and to exploit them into a solving method.

As it will expose in the article, the concept of super-jobs has several interests. It enables:

- First, to reduce the combinatorics of the problem and to make feasible the use of efficient methods developed for small size problems.
- Secondly, as for the Variable Neighborhood Search strategy, to modify the landscape during the search and so, to easier escape from local optima.

Experiments show that exploiting these super-jobs allows to obtain very good results as a large number of new best solutions have been found for large size instances of the commonly used Taillard's benchmark.

To expose the way we propose to define and exploit super-jobs, this article is organized as follows: Section 2 presents the No-Wait Flowshop Scheduling Problem and a very brief literature review on the problem. Then, Section 3 gives the analysis we conducted and our definition of super-jobs. Our approach to exploit super-jobs is described in Section 4 and experiments are conducted and discussed. The last section presents some conclusions as well as some perspectives.

2 No-Wait Flowshop Scheduling Problem

2.1 Problem Description

Presentation of the problem The No-Wait Flowshop Scheduling Problem (NWFSP) is a variant of the well-known Permutation Flowshop Scheduling Problem (PFSP), where no waiting time is allowed between the processing of a job on the successive machines [13]. Despite this constraint, the NWFSP remains a NP-hard problem. More formally, the NWFSP 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 the *processing time* of a job i on a machine j. As for the PFSP, the sequence of jobs is the same on each machine, hence, a solution of the NWFSP 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.

No-Wait variant specificity The NWFSP possesses a characteristic not present in the classical PFSP that enables to reduce the computation time of the makespan of a sequence. Indeed, the delay between two jobs *i.e.,* the start-up interval between two defined consecutive jobs of a sequence on the first machine, is constant and does not depend on their position within the sequence [2]. Let $d_{i,i'}$ be the delay of two jobs i and i', it is computed as follows:

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

Then, the completion time $C_i(\pi)$ of the job π_i of sequence π can be directly computed from the delays of the preceding jobs as follows:

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

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

Neighborhood operator Approaches we will present, involve local search methods. These ones use a neighborhood operator to move from a solution to another. In this work, we use a permutation representation and the *insertion operator* [16], as it is known to make local search more efficient on flowshop problems [6]. This operator selects within a sequence π, a job π_i and inserts it at position k ($i \neq k$). Hence, jobs between positions i and k are shifted. Two sequences π and π' are said to be neighbors when they differ from exactly one insertion move. The size of the neighborhood induced by this operator *i.e.,* the number of neighboring solutions, is $(n-1)^2$. 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 $O(1)$ [10].

2.2 State-of-the-art

Optimization approaches Many heuristics and metaheuristics have been proposed to solve scheduling problems and in particular the No-Wait Flowshop variant.

Heuristics are either adaptations of heuristics developed for the classical permutation flowshop problem or have been specifically designed for the NWFSP. Hence, the well-known constructive heuristic NEH (Nawaz, Enscore, Ham [8]), initially designed for the classical permutation flowshop, has been successfully applied on the No-Wait variant. Among heuristics specifically designed for the NWFSP, we may cite BIH (Bianco et al. [3]), BH (Bertolissi [2]) and RAJ (Rajendran [12]).

Contrary to constructive heuristics, metaheuristics are able to find solutions with a higher quality. Both bio-inspired and local search algorithms have been proposed to tackle the NWFSP, such as genetic algorithms [1], particle swarm optimization [9], differential evolution [11] as well as tabu search [5, 15] and simulated annealing [1]. Recently, Ding et al. [4] proposed a very efficient approach named TMIIG (Tabu-Mechanism Improved Iterated Greedy) based on a variable neighborhood search (VNS) [7]. In the perturbation phase, the authors have chosen to use the efficient destruction-construction method of the Iterated Greedy (IG) [14] and add a tabu mechanism to avoid scheduling a job at its previous positions during the different destruction-construction phases.

Benchmark Taillard's benchmark [17], initially provided for the PFSP, is also widely used in the literature for the NWFSP. This benchmark proposes 120 instances, organized by 10 instances of 12 different sizes with a number of jobs $N \in \{20, 50, 100, 200, 500\}$ and $M \in \{5, 10, 20\}$. The higher the number of jobs and/or the number of machines, the more difficult the instance to solve. Data provided by these instances is the processing time of each job on each machine. Taillard's instances are said to be random as the processing times are uniformly generated according to $\mathcal{U}[1; 99]$. As far as we know, TMIIG is one of the best algorithms to solve Taillard's instances since it has recently (in 2015) found new best solutions for the largest instances.

3 Super-jobs: Promising Sub-Sequences of Consecutive Jobs

For the NWFSP, each job is processing without interruption between the successive machines. Therefore, a question is: does this specificity lead to a particular structure of the best solutions of a given instance. In this section, we conduct an analysis of the global optimum and several local optima solutions in order to extract structural information on them. This leads us to define a promising sub-sequence of consecutive jobs as a *super-job*. Then, we present a methodology to identify *super-jobs* of an unknown instance in order to use them to solve it.

3.1 Structural Analysis of Optima Solutions

This analysis aims at extracting similarities in the structure of good quality solutions. We conduct this analysis on *small* instances (low number of jobs) with processing times uniformly generated following the methodology of Taillard's instances (see Section 2.2). We report here, as an example, the analysis of an instance with 12 jobs and 5 machines. This problem size (12) enables to exhaustively enumerate the search space and therefore, to identify the global optimum and the best local optima[1]. Indeed, local optima are interesting to analyze since they may trap local search methods that explore the search space moving iteratively to improving neighbors. In the following, the term optimum solutions (or optima) is used to deal with the global or local optimum solutions more generally.

Figure 1 gives the global optimum and the 10 best local optima of the studied instance of size 12. For this small instance, it is easy to see that these best optima share a similar structure. Indeed, job 8 is always positioned at the beginning of the schedule and three sub-sequences of two consecutive jobs are present in all of them: [9 1] colored in blue, [0 4] in green, and [11 2] in red.

Solutions reported in Figure 1 are, except for the first one, local optima. This means that applying the insertion operator cannot improve them. However, if we consider each identified sub-sequence of consecutive jobs as a unique job, the first local optimum ($C_{max} = 1036$) differs from the global optimum ($C_{max} = 1021$) by the move of the sub-sequence [11 2] only. In the same manner, applying the insertion operator on the other local optima with the consideration of the identified sub-sequences instead of single jobs allows to move all the local optima (except the one with $C_{max} = 1176$) to the global optimum. This observation motivates the replacement of original jobs by promising sub-sequences of jobs to help to reach better quality solutions.

[1]Local optima are solutions that have no better neighboring solutions *i.e.,* no insertion move could lead to a strictly better quality solution.

C_{max}	Solution											
Global Optimum: 1021	8	3	7	5	10	[9	1]	[0	4]	6	[11	2]
Local Optima: 1036	8	3	7	[11	2]	5	10	[9	1]	[0	4]	6
1075	8	10	[9	1]	[0	4]	7	5	3	[11	2]	6
1090	8	3	5	10	6	[0	4]	7	[11	2]	[9	1]
1103	8	3	5	10	6	[0	4]	7	[9	1]	[11	2]
1132	8	10	6	[0	4]	7	5	3	[9	1]	[11	2]
1132	8	3	7	[11	2]	5	10	9	6	[0	4]	1
1176	8	3	5	10	[9	1]	[11	2]	7	6	[0	4]
1189	8	10	6	[0	4]	7	5	3	[11	2]	[9	1]
1232	8	10	[9	1]	[0	4]	7	3	5	6	[11	2]
1246	8	3	7	10	[9	1]	[11	2]	5	6	[0	4]

Figure 1: Quality (C_{max}) and description of the global optimum and the 10 best local optima for the studied instance of size 12. Different sub-sequences of consecutive jobs are colored.

In this studied (small) example, only sub-sequences of two consecutive jobs were found. However, the size is not limited. Hence, if a job a is always followed by a job b and, the job b is always followed by a job c then, the sub-sequence $[a\ b\ c]$ of three consecutive jobs has to be considered rather than the two sub-sequences $[a\ b]$ and $[b\ c]$ separately.

The advantages of considering sub-sequences of consecutive jobs as unique jobs are several. First, for a complexity point of view, it will reduce the combinatorics of the problem *i.e.,* the number of potential solutions. Secondly, for a local search point of view, this will modify the search space and the landscape induced by the insertion operator and so, new regions of the search space may be reachable.

Therefore, as these sub-sequences may be a good opportunity to solve large problems, the question is: how to define and identify them?

3.2 Definition of Super-Jobs

The previous exhaustive analysis of the structure of local optima for small size instances leads us to suppose that a similar behavior appears on larger ones. In this section, we present the methodology we propose to identify promising sub-sequences of an unknown instance to be solved.

Regardless the size of the search space, good quality solutions and more precisely, good quality local optima, hopefully share a similar structure. When the search space is non enumerable, it is commonly admitted to use a sample of solutions in order to make analyses of solutions, characteristics . . .

Here, we propose to extract the promising sub-sequences from a pool $\mathcal{P}^*$ of good quality local optima and to define a *super-job* with a confidence of σ, any sub-sequence of consecutive jobs that appears at least σ percent of times in solutions of $\mathcal{P}^*$. For example, if $\sigma = 50\%$, the super-jobs are the sub-sequences of consecutive jobs shared by half of the solutions of $\mathcal{P}^*$. Let us note that only the longest sub-sequences are considered as super-jobs. For example, if both $[a\ b]$ and $[b\ c]$ appear at least σ percent of times in $\mathcal{P}^*$, only $[a\ b\ c]$ is defined as a super-job.

This methodology has the advantage to be relevant regardless the problem size. However, the main drawback may be the computational time required to generate the pool of good quality solutions. Therefore, the size of the pool has to be fixed carefully: too large means that too much time would be spent to generate the pool, too small means that the identification of the super-jobs would be insignificant.

4 Super-jobs to improve the performance of the Iterated Greedy

Super-jobs have been defined as common structural characteristics of good quality solutions. In this section we propose an approach that exploits these super-jobs to improve an already efficient heuristic – the Iterated Greedy – in order to reach new best solutions for the Taillard's instances.

4.1 Iterated Greedy Algorithm

The Iterated Greedy (IG) algorithm [14], initially proposed for the PFSP, is an iterated local search, based on the insertion operator, whose perturbation phase removes some jobs from a solution, and reinserts them one by one at their best position *i.e.,* the position that minimizes the partial makespan. The local search is an iterative improvement: each job of the sequence is considered in a random order and is re-inserted at its best position. This process is repeated until a local optimum is reached. The acceptance criterion of IG is inspired from the one of the simulated annealing and, checks if the new local optimum found is better or not than the best one ever found during the run. IG is known to be efficient to solve many variants of PFSP. However, even if it is able to reach good quality solutions in a reasonable computational time, it is not able to reach, for the NWFSP, the best-known solutions of the larger instances of Taillard's. Indeed, currently, the best algorithm for the NWFSP is the recent algorithm TMIIG [4], inspired from IG (see Section 2.2). However, since the performance of IG is doubtless to solve small and medium sizes instances and, since the use of super-jobs decreases the problem size, we propose to take advantage of both IG and the super-jobs.

4.2 Proposed Approach

```
Algorithm 1: Proposed Approach
  Data: $\mathcal{P}^*$: pool of solutions; $\Sigma = \sigma_1, \sigma_2, \ldots$: list of confidence levels in increasing order;
        SJ: list of super-jobs; $\pi$: solution.
  SJ = identify($\mathcal{P}^*, \sigma_1$) ;          /* Identifies Super-jobs with a confidence $\sigma_1$ */
  $\pi$ = init(SJ) ;                                  /* Initializes $\pi$ with identified SJ */
  foreach $\sigma$ in $\Sigma$ do
      SJ = identify($\mathcal{P}^*, \sigma$);                     /* Not executed for $\sigma_1$ */
      $\pi$ = IG($\pi$, SJ);                       /* Runs IG from $\pi$ with identified SJ */
  return $\pi$
```

The proposed approach, described in Algorithm 1, aims at identifying super-jobs of several increasing levels of confidence σ during the search. Given a pool of good quality local optima, whose generation will be discussed later, the proposed approach first identifies super-jobs regarding the first level of confidence. Afterwards, an initialization method generates a first solution with these identified super-jobs. A process is then, iterated for each value of level of confidence selected, alternating between a phase of super-jobs identification and a phase of improvement with IG. The Iterated Greedy algorithm is executed on the solution considering super-jobs as jobs of the problem. As IG has no natural stopping criterion, a maximal time, as well as a maximal number of iterations without improvement, are used to stop the IG phase. Once all the levels of confidence are used, the algorithm returns the best-found solution over the run.

Table 1 explains the dynamic of the approach through an example on the instance `ta023` of Taillard (20 jobs, 20 machines), using a pool of solutions $\mathcal{P}^*$ of size 10 and the list of confidence levels $\Sigma = \{60\%, 70\%, \infty\}$ – where $\sigma = \infty$ means that no super-job is created (the problem is solved with original jobs) –. In this example, during the first phase ($\sigma = 60\%$), seven super-jobs are identified: one of size 12, two of size 2 and the forth-remaining ones of size 1. Therefore, the problem size is decreased from 20 to 7. The initialization method then builds a local optimum with a quality of 3021 that IG is not able to improve. Then, considering a confidence level of 70%, some super-jobs are decomposed. Indeed,

σ	Phase	Problem size	C_{max}	Solution π
60	Identify	7	-	**[1 7]** 2 **[3 19]** 5 11 **[13 15 14 17 9 4 8 18 0 12 6 10]** 16
	Init		3021	**[3 19]** 16 5 2 11 **[13 15 14 17 9 4 8 18 0 12 6 10]** **[1 7]**
	IG		3021	**[3 19]** 16 5 2 11 **[13 15 14 17 9 4 8 18 0 12 6 10]** **[1 7]**
70	Identify	14	3021	**[3 19]** 16 5 2 11 **[13 15]** 14 17 **[9 4]** 8 **[18 0]** 12 **[6 10]** **[1 7]**
	IG		3013	**[3 19]** 16 5 2 12 17 **[18 0]** 11 **[13 15]** 14 **[9 4]** 8 **[6 10]** **[1 7]**
∞	Identify	20	3013	3 19 16 5 2 12 17 18 0 11 13 15 14 9 4 8 6 10 1 7
	IG		3013	3 19 16 5 2 12 17 18 0 11 13 15 14 9 4 8 6 10 1 7

Table 1: Our proposed approach on instance `ta023` of Taillard with $\sigma = \{60\%; 70\%; \infty\}$

the largest super-job is decomposed into eight smaller ones. The problem size is now equal to 14. IG manages to find a better solution (quality 3013). It appears, in this case, that the global optimum is reached within this second phase. Otherwise, the last phase ($\sigma = \infty$) could still lead to an improving solution.

4.3 Experiments

Experimental Protocol Each execution of our approach on a given instance $\mathcal{I}$ returns a solution π. To measure the quality of the execution, the Relative Percentage Deviation (RPD) is computed from the best-known solution π^* of $\mathcal{I}$ as follows:

$$RPD = \frac{C_{max}(\pi) - C_{max}(\pi^*)}{C_{max}(\pi^*)} * 100 \tag{3}$$

Our approach is stochastic hence, 30 runs are required to make robust the experimental results. The performance of our approach for an instance $\mathcal{I}$ is the average of the 30 RPD computed.

As exposed in section 2.2, the benchmark used to evaluate the performance of the proposed method is composed of Taillard's instances [17] organized by 10 instances of 12 different sizes. The algorithm is implemented using C++ and the experiments were executed on an Intel(R) Xeon(R) 3.5GHz processor.

Following a preliminary study, several elements have been settled for these experiments:

- **Generation of the pool of solutions:** 10 executions of IG during $n^2 * 10ms$, is enough to generate a pertinent pool of solutions that enables to extract knowledge ;
- **Determination of the list of levels of confidence:** using three different levels of confidence $\{60\%, 70\%, \infty\}$ allows to obtain very good results (even if 60% and 70% seem to be very close one to each other, super-jobs identified are different)
- **Procedure** $Init$**:** The designed initialization procedure (used at phase 1 for σ_1) is inspired from NEH [8] hybridized with a stochastic local search to reach local optima.
- **Stopping criterion of IG:** At each phase, a maximal time of $n^2 * 10ms$ (where n is the number of super-jobs of the phase), and a maximal number of iterations without improvement of 100 are defined.

Experimental Results To analyze performances of the method, results obtained are compared with the best-known solutions of the literature, reported in [4]. Figure 2 indicates, for each instance (10 instances per size) if the best-known solution of the literature has been reached (cells colored in gray) and if this best-known has been improved (non empty cell). Hence when a number is present in a cell, this indicates the number of times the method overtaken the previous best-known solution over the 30 executions – an 'x' indicates 30/30 –. This Figure shows that for all instances the method is able to reach the best-known of the literature, and in addition it shows that all instances with 100, 200 and 500 jobs have been

Jobs / Machines	20										50										100										200										500									
5	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	17	8	▨	21	▨	▨	14	▨	X	19	14	X	23	21	29	25	14	15																				
10	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	24	21	20	X	25	X	23	X	29	25	X	X	X	X	X	X	X	X	X	X										
20	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	14	28	20	25	26	X	29	28	27	28	X	X	X	X	X	X	X	X	X	X	X	X	X	29	X	25	X	X	X	X

▨ Best known of literature   ▨ # X New best solution

Figure 2: Results on Taillard's instances (organized by size).

050×5		100×5		100×10		100×20		200×10		200×20		500×20	
Instance	Best	Instance	Best	Instance	Best	Instance	Best	Instance	Best	Instance	Best	Instance	Best
Ta31	3,161	Ta61	**6,366**	Ta71	**8,061**	Ta81	**10,685**	Ta91	**15,264**	Ta101	**19,586**	Ta111	**46,445**
Ta32	3,432	Ta62	**6,221**	Ta72	**7,862**	Ta82	**10,569**	Ta92	**15,044**	Ta102	**19,980**	Ta112	**46,921**
Ta33	**3,210**	Ta63	**6,108**	Ta73	**8,017**	Ta83	**10,600**	Ta93	**15,285**	Ta103	**19,818**	Ta113	**46,332**
Ta34	**3,338**	Ta64	**6,001**	Ta74	**8,332**	Ta84	**10,590**	Ta94	**15,137**	Ta104	**19,796**	Ta114	**46,736**
Ta35	3,356	Ta65	**6,188**	Ta75	**7,939**	Ta85	**10,510**	Ta95	**15,126**	Ta105	**19,751**	Ta115	**46,541**
Ta36	**3,346**	Ta66	**6,058**	Ta76	**7,773**	Ta86	**10,633**	Ta96	**15,022**	Ta106	**19,856**	Ta116	**46,837**
Ta37	3,231	Ta67	**6,228**	Ta77	**7,851**	Ta87	**10,793**	Ta97	**15,324**	Ta107	**19,998**	Ta117	**46,346**
Ta38	3,235	Ta68	**6,115**	Ta78	**7,881**	Ta88	**10,810**	Ta98	**15,168**	Ta108	**18,927**	Ta118	**46,697**
Ta39	**3,070**	Ta69	**6,360**	Ta79	**8,140**	Ta89	**10,705**	Ta99	**15,024**	Ta109	**19,831**	Ta119	**46,566**
Ta40	3,317	Ta70	**6,372**	Ta80	**8,095**	Ta90	**10,765**	Ta100	**15,273**	Ta110	**19,794**	Ta120	**46,624**

Table 2: Best known solutions of Taillard's instances. A bold value indicates a new best solution has been found out by our approach.

improved with the proposed approach. In addition, Table 2 gives for each instance the quality of the new best solution obtained[2].

Analysis of the Results Table 3 reports some information to deeper analyze the behavior of the method in the different phases ($Init$, IG) according to the level of confidence. Left part of this Table reports the size of the problem *i.e.,* the number of jobs for each phase. These jobs are either original jobs, or the super-jobs constructed by the concatenation of several jobs, as explained before. This measure gives the combinatorics of the problem. Hence, for example, for instances of size 200, when $\sigma = 60\%$, the number of jobs is around 80-90. This means that the size of the problem has been divided by more than 2. In the following phase, when $\sigma = 70\%$, some super-jobs are decomposed and the number of jobs is around 130-140. The combinatorics is still reduced. This analysis, also shows that there is a real difference between the two levels of confidence, as the number of jobs to consider is different. This Table also shows that the number of machines has an impact on the identification of super-jobs. Indeed for a given number of jobs, the more the number of machines, the more reduced the combinatorics. As far as instances of size 500 are concerned, the high level of combinatorics still present may be explained by the use of IG to generate the pool of solutions. Indeed, IG has difficulty to converge within the time allowed. Hence the solutions of the pool are too diversified and it is difficult to extract enough pertinent sub-sequences of jobs.

The middle and right part of the Table enables to study the convergence of the approach. The middle part indicates for each phase, the number of times (over the 30 executions) the method – $Init$ or IG – reached the best solution (average of the 10 instances of each size). We can observe, that for instances of size 20, the best solution is mainly reached during the initialization phase. Indeed, for small size instances, IG is very efficient and manages to reach best-known solutions. Up to size 100, the three first phases are sufficient to reach very good solutions. The super-jobs identified with $\sigma = 70\%$ are pertinent and, the combinatorics has been highly decreased. For larger size problems, some improvements are still obtained in the last phase when the original problem is considered. We could wonder whether an intermediate phase with $\sigma = 80\%$ could help to find out better or best solutions.

This analysis is re-enforced by the right part of the Table that reports the average RPD at each phase.

[2]Best solutions of Taillard's instances are reported in http://lucienmousin.fr/nwfs

Instances	Problem size		End of convergence				RPD value			
	60%	70%	$\text{Init}_{60\%}$	$\text{IG}_{60\%}$	$\text{IG}_{70\%}$	IG_{∞}	$\text{Init}_{60\%}$	$\text{IG}_{60\%}$	$\text{IG}_{70\%}$	IG_{∞}
20×5	1.20	10.19	29.4	0	0.6	0	0.043	0.043	0	0
20×10	1	10	30	0	0	0	0	0	0	0
20×20	1.21	10.23	29.1	0.5	0.4	0	0.008	0.005	0.001	0.001
50×5	9.39	29.22	3.7	5.2	19.3	1.8	0.645	0.317	0.056	0.049
50×10	6.15	28.10	7.5	3.5	15.4	3.6	0.545	0.422	0.077	0.060
50×20	4.23	26.90	17.7	0.9	6.6	4.8	0.372	0.315	0.060	0.044
100×5	45.48	71.50	0	14.9	8.6	6.5	1.825	0.201	0.176	0.164
100×10	29.53	63.93	0	4.2	16.3	9.5	1.189	0.302	0.166	0.146
100×20	27.16	63.06	0	2.3	13.1	14.6	1.037	0.370	0.164	0.136
200×10	95.73	144.55	0	4.9	12.5	12.6	2.004	0.224	0.184	0.167
200×20	82.17	137.99	0	2.2	13	14.8	1.616	0.289	0.200	0.180
500×20	270.34	377.25	0	0.7	7.7	21.6	1.971	0.218	0.168	0.154

Table 3: Phases analysis. Left: problem size (considering the super-jobs identified). Middle: number of times the methods end at the given phase. Right: average RPD at each phase.

Thus it indicates, how far from the new best (or best-known) solution, are solutions reached after each phase. For large instances we can observe the high improvement between the initialization and the phase with $\sigma = 60\%$. In the following phases, even if some better solutions are reached, the improvement is less important but significant.

4.4 Discussion

The experimental results proved the performance of our approach since new best solutions have been found out for every largest Taillard's instances. This section provides a discussion on two important aspects of the method: the computational time required for the generation of the pool of solutions, and the analysis of the dynamicity of the method.

Discussion on the generation of the pool $\mathcal{P}^*$ The computational time of the proposed method may represent a drawback when solving large instances. This computational time is partly explicated by the generation of the pool of solutions $\mathcal{P}^*$. Indeed, for example, to solve a 200 jobs instance, 400 seconds are required to generate one solution of the pool, hence around one hour for the 10 solutions of the pool. This is quite long, but not so important if we want to obtain new best solutions.

The fact is that the performance of the approach is based on the knowledge extraction from this pool of initial solutions used to identify pertinent super-jobs. Preliminary results shows that the quality of the solutions of the pool impacts a lot the performance, and constructive heuristics does not give enough good quality solutions to identify reliable super-jobs. Hence, we chose IG, with a time limit, to give pretty good quality solutions for the pool of solutions, but it is time consuming. In addition, we tested different sizes for the pool. Indeed, the higher the size, the larger the computational time to generate the pool whereas the lower the size, the lower the chance to have a representative pool. Following these tests we decided to generate a pool of 10 solutions only, as pertinent super-jobs can be found out, even with so few solutions. A small pool enables to reduce a lot the whole computational time of the approach.

Since our approach is stochastic, in the exposed experiments, the performance is evaluated from 30 executions for each instance. Each execution generates its own pool of good quality solutions. We drove some parallel experiments where a single pool of 10 solutions was generated only and was shared between the 30 executions. The experimental results were similar: best-known solutions of the literature were reached for the smallest instances of Taillard, and improved for the largest one. Using such a shared pool allows decreasing the whole computational time for the 30 executions.

During the analysis of the method, we also noticed that for largest instances (with 500 jobs, mainly) a better pool of solutions improves the identification of super-jobs and then still reduces the combinatorics.

Thus, we can imagine leaving more computational time to IG to generate a shared pool of better quality solutions, instead of generating one different pool for each execution.

Discussion on the behavior of the method Another interesting aspect is that he performance of the approach lies on the reduction of the combinatorics of the problem made possible by the particular structure of the best quality local optima. In the search space, each local optimum is the 'center' of a basin of attraction. All the basins make the landscape very rugged for local search methods. The perturbation phase in the Iterated Greedy has been designed to escape from local optima, and so, from their attraction basins. However, the basins of attraction are not side-by-side but included in each other; The best local optimum of a large basin may be the center of other ones. Hence, even with a perturbation, a local search method often remains in the same large basin of attraction he started. The reduction of the combinatorics of the problem, with the identification of super-jobs, produces an interesting effect on the landscape. Indeed, original local optima do not exist in the reduced landscape and so, for its attraction basin. Therefore, regions of the landscape are smoothed and original basins of attractions get larger and, the performance of IG at each iteration of our approach are explained. For example, for instances with 20 jobs (the easiest of the Taillard instances), IG ends to converge close to the best-known solutions without reaching it, whereas with the reduction of the combinatorics, it reaches it each time. The reduction of the number of attraction basins helps IG to move towards the best one. Exploiting super-jobs erases rugged regions of the search space and, enables to increase the performance of IG.

5 Conclusion

This paper presents a new approach based on the Iterated Greedy algorithm that has found out 64 new best solutions of Taillard's instances for the No-Wait Permutation Flowshop Scheduling Problem (NWFSP). The novelty of this approach is to modify the landscape during the search by first reducing the combinatorics of the initial problem and then, increasing it. This process is allowed by the observed common structure of the best good quality solutions of the NWFSP that present common sub-sequences of consecutive jobs. The identification of these sub-sequences has been discussed in the paper and has led to the definition of super-jobs with a confidence level. Super-jobs are considered as a single job of the problem therefore, the confidence level indirectly determines the size of the problem to be solved. The proposed approach consists in exploiting these super-jobs and executing the Iterated Greedy iteratively.

In the experiments, we show that super-jobs, identified with a middle level of confidence (here, 60%) allow reducing by the half the combinatorics of the problem. Therefore, the Iterated Greedy algorithm more easily identifies good regions of the search space. The increase of the confidence level decomposes little by little the previous identified super-jobs hence, the search space is revealed gradually. The performance of this approach shows the benefit of using knowledge into optimization method such as meta-heuristics.

Future works will focus on how to adapt this approach for other permutation problems and so, how to identify other means to extract knowledge from the best solutions. Moreover, the different increasing levels of confidence, needed to identify the super-jobs, will be analyzed to try to set the values in function of the instance.

References

[1] Tariq Aldowaisan and Ali Allahverdi. New heuristics for no-wait flowshops to minimize makespan. *Computers & Operations Research*, 30(8):1219–1231, jul 2003.

[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] Jian-Ya Ding, Shiji Song, Jatinder N.D. Gupta, Rui Zhang, Raymond Chiong, and Cheng Wu. An improved iterated greedy algorithm with a tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Applied Soft Computing*, 30:604–613, may 2015.

[5] Józef Grabowski and Jarosław Pempera. Some local search algorithms for no-wait flow-shop problem with makespan criterion. *Computers & Operations Research*, 32(8):2197–2212, aug 2005.

[6] Panos Kouvelis, Richard L. Daniels, and George Vairaktarakis. *IIE Transactions*, 32(5):421–432, 2000.

[7] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, nov 1997.

[8] 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.

[9] Quan-Ke Pan, M. Fatih Tasgetiren, and Yun-Chia Liang. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, 35(9):2807–2839, sep 2008.

[10] 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.

[11] B. Qian, L. Wang, R. Hu, D.X. Huang, and X. Wang. A DE-based approach to no-wait flow-shop scheduling. *Computers & Industrial Engineering*, 57(3):787–805, oct 2009.

[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] Rubén Ruiz and Thomas Sttzle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, mar 2007.

[15] Hamed Samarghandi and Tarek Y. ElMekkawy. A meta-heuristic approach for solving the no-wait flow-shop problem. *International Journal of Production Research*, 50(24):7313–7326, dec 2012.

[16] T. Schiavinotto and T. Stützle. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34:3143–3153, 2007.

[17] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, jan 1993.