Minimizing Total Tardiness in a Hybrid Flexible Flowshop with Sequence Dependent Setup Times

Aymen Sioud, Caroline Gagné, and Marc Gravel
Département d'informatique et de mathématique
Université du Québec à Chicoutimi
Email: asioud@uqac.ca, c3gagne@uqac.ca, mgravel@uqac.ca

***Abstract*—In this paper, we propose different local search algorithms to solve a realistic variant of the flowshop problem. The variant considered here is a hybrid flexible flowshop problem with sequence-dependent setup times, and with the objective of minimizing the total tardiness. In this variant of flowshop, stage skipping might occur, i.e., not all stages must be visited by all jobs. This scheduling problem is frequently used in batch production, helping to reduce the gap between research and operational use. While there is some research on minimizing the makespan, to our knowledge no work has been reported on minimizing the total tardiness for this problem. The proposed approaches present different neighborhood searches. Numerical experiments compare the performance of the different algorithms on new benchmarks generated for this problem.**

***Keywords*—hybrid flexible flowshop; sequence dependent setup times; total tardiness; local search; scheduling.**

I. INTRODUCTION

Scheduling problems are decision-making processes with the goal of optimizing one or more objectives. Indeed, they deal with the allocation of resources to perform a set of activities over a period of time [1]. The Flowshop scheduling problem is one of the most well-studied scheduling environments in the last decades. In this configuration, all jobs follow the same operational order, where a set of *n* jobs need to be processed in a set of *m* machines disposed in series. All jobs follow the same processing route through the machines, i.e., they are first processed on machine 1, then on machine 2 and so on until machine *m*. This associated problem can be considered as the basic model for several variants of real problems. Moreover, real production systems rarely employ a single machine at each stage. Therefore, in many situations, the regular flowshop problem is extended to a set of usually identical parallel machines at each stage. That is, instead of having a series of machines, we have a series of stages of parallel machines. The goal here is to increase the capacity and reduce the impact of bottleneck stages. Furthermore, it is frequent in practice that optional treatments are applied on products, like polishing or additional decorations in ceramic manufacturing as examples [2][3]. In this latter case, some jobs will skip some stages. We obtain thereby the hybrid flexible flowshop.

Moreover, in many industries such as pharmaceutics, metallurgy, ceramics and automotive manufacturing, there are setup times on equipment between two different jobs. Many papers assume that setup time is negligible, or part of the job processing time. But explicit setup times must be included in scheduling decisions in order to model a more realistic variant of hybrid flowshop scheduling problems. These setup times may or may not be sequence dependent. Dudek et al. [4] reported that 70% of industrial activities include dependent setup times. More recently, Conner [5] pointed out in 250 industrial projects that 50% of these projects contain setup-dependent times and when these setup times are applied, 92% of the order deadline could be met. Production of good schedules often relies on good management of these setup times [6][7].

The present paper considers the hybrid flexible flowshop problem with sequence dependent setup times (SDST/HFFS) minimizing the total tardiness. In accordance with the notation for hybrid flowshops introduced by Vignier et al. [8] who extended the well-known three fields notation $\alpha/\beta/\gamma$ of Graham et al. [9], this problem is noted as $((PM)^{(i)})_{i=1}^{m}/F_j, s_{ijk}/\Sigma T_j$. According to Du and Leung [10], the total tardiness minimization problem is NP-hard for the specific case of one machine and therefore the SDST/HFFS total tardiness problem studied in this article is also NP-hard.

The $((PM)^{(i)})_{i=1}^{m}/F_j, s_{ijk}/\Sigma T_j$ problem may be defined as a set of *N* jobs, *N*={*1,..., n*} available for processing at time zero on a set of *M* stages, *M*={*1,..., m*}. At every stage *i*, $i \in M$, we have a set of M_i, M_i={*1,..., m_i*} of identical parallel machines. Every machine at each stage can process all the jobs. Each job has to be processed in exactly one out of the M_i identical parallel machines at stage *i*. However, some jobs will skip some stages. F_j denotes the set of stages that the job *j*, $j \in N$ has to visit. Furthermore, only stage skipping is allowed, so it is not possible for a given job to visit stages {1, 2, 3} and another one to visit stages {3, 2, 1}. p_{ij} denotes the processing time of job *j* at stage *i*. Each job have a due date noted as d_j. Finally, s_{ijk} denotes the setup time between jobs *j* and *k*, $k \in N$ at stage *i*. The completion time of a given job *j* at stage *i*, noted as C_{ij}, is calculated as $C_{ij} = max\{C_{i,j-1}; C_{i-1,j}\} + s_{i,j-1,j} + p_{ij}$, where $C_{i,j-1}$ is the completion time of the previous job in the sequence that was assigned to the same machine and $C_{i-1,j}$ is the completion time of job *j* at the previous stage that this job visited. The tardiness of a job *j* noted as T_j is calculated as $T_j = max\{0, C_{mj} - d_j\}$ where C_{mj} is the completion time of the job *j* on the last stage. The optimization criterion is the minimization of the total tardiness of all the jobs, which will be expressed as $\sum_{j=1}^{n} T_j$.

This problem is a significant one in real production cases commonly found in various types of manufacturing systems, such as ceramics [11] and the production of printed circuit boards [12]. In addition, the total tardiness minimization is an optimization criterion of strategic importance. Indeed,

 ISBN: 978-1-61208-365-0

Allahverdi et al. [6] highlight the consequences that order tardiness may present, such as the cost of lost sales and even penalties in a sales contract. To the best of our knowledge, there has been no study on the SDST/HFFS problem minimizing the total tardiness, but we can find in the literature some papers proposing heuristics and/or metaheuristics for the SDST/HFFS problem minimizing the makespan (C_{max}), such as [13]–[17], and studies about related problems, such as [2][11][18]–[20], also minimizing the makespan.

Concerning the total tardiness minimization, there are papers about variations of the SDST/HFFS problem. Indeed, Ruiz et al. [21] introduce two iterated greedy heuristics (IGH) for a hybrid flowshop problem with the objectives of minimizing the makespan and the weighted tardiness. In this paper, we consider release dates for machines, machine eligibility, possibility of the setup times to be both anticipatory and non-anticipatory, precedence constraints and time lags. Mainieri et al. [22] propose five heuristics to solve a hybrid flowshop without setup times. The heuristics are based on list scheduling algorithms. Naderi et al. [18] present a simulated annealing to solve a hybrid flowshop with setup-dependent and transportation times. They used the well known NEH heuristic [23] and three different neighborhood structures based on the swap, shift and inversion operators. Jungwattanakit et al. [24] consider the flexible flowshop problem with both sequence- and machine-dependent setup times. They propose constructive heuristics, shift neighborhood search and a genetic algorithm. Finally, we refer the reader to a detailed literature revue about the hybrid flowshop in [25], where the authors highlight the lack of studies dealing with the setup times and the total tardiness in general.

In this work, to solve the SDST/HFFS problem, we introduce different local search algorithms tested on a new benchmark. The body of this paper is organized into five sections. Section II describes the proposed resolution approaches. The computational testing, benchmark and discussion are presented in Section III. Finally, we conclude with some remarks and future research directions.

II. HEURISTICS FOR THE SDST/HFFS TOTAL TARDINESS PROBLEM

Heuristic algorithms can be classified into dispatching rules, constructive and improvement heuristics. Dispatching rules, also called greedy algorithms, are algorithms for which the decision about which job to scheduled next is made based on the jobs or/and the time in greedy way. Constructive heuristics build a schedule from empty or partial solution by making a series of passes through the list of unscheduled jobs, where at each pass one or more jobs are selected and scheduled. In this case, the selection of the job to be scheduled can be done using the history of jobs running, or by computing tentative schedules. Contrary to constructive heuristics, improvement heuristics start from an existing solution and apply some improvement procedure. We present in the following subsections the retained dispatching, constructive and improvement heuristics.

A. Dispatching rules and Constructive heuristics

According to Pinedo [1], dispatching rules are used very often in practice and as an initial sequence in some improvement heuristic and metaheuristic methods. They are useful when one attempts to find a reasonably good schedule. Let $C_j(S)$ be the completion time of job $j \notin S$ if it is scheduled at the end of a sequence S. We consider in this paper :

- Early Due Date (EDD) : At time *t*, the job with minimum d_j value is selected.
- SLACK : At time *t*, the job with the minimum value of $d_j - C_j(S)$ is selected.
- Modified due date (MDD): At time *t*, the job with the minimum value of $max\{d_j - C_j(S)\}$ is selected.

Pinedo [1] mentions other dispatching rules, but states that the three aforementioned rules are common for total tardiness minimization.

Concerning constructive heuristics, the NEH heuristic [23] is regarded as the best heuristic for the permutation flowshop scheduling problem with the makespan minimization criterion [2]. In the original version of NEH, the jobs are sorted in non-increasing order of the sum of processing times on all machines. In the presence of due dates, several methods can be used to sort the jobs, such as the EDD or SLACK rules.

B. Improvement heuristics

An improvement heuristic algorithm starts with a solution and iteratively tries to obtain a better solution. Neighborhood search algorithms (alternatively called local search algorithms) are a wide class of improvement heuristics where at each iteration an improved solution is found by searching the neighborhood of the current solution. A comprehensive survey on neighborhood search algorithms (NSAs) is given by Prandtstetter and Raidl [26]. An outline of the local search minimization algorithm is given in Figure 1 where the evaluation function f is the total tardiness in our case. As we can see from a $Current$ solution we generate a neighborhood $\mathcal{N}(Current)$ of feasible solutions achievable from $Current$. Then, from this generated neighborhood, we select a solution which will become the $Current$ solution. The choice of the neighborhood structure and the selection scheme will be very important in finding a good solution [26]. We describe in the next subsections a variety of proposed strategies to solve the SDST/HFFS total tardiness problem.

Require: $Current$, an initial solution
Require: $Next$, a solution
  repeat
    Generate $\mathcal{N}(Current)$ a neighborhood of $Current$
    $Next \leftarrow$ Select a solution $\in \mathcal{N}(Current)$
    if $f(Next) \leq f(Current)$ **then**
      $Current \leftarrow next$
    end if
  until Achieving a stopping criterion

Fig. 1. Local Search minimization algorithm

1) Neighborhood search structure: Several neighborhood search structures have been applied to scheduling problems [26]. In this section, we introduce some of them that have been extracted from published works and used in this paper.

Swap move: The positions of two randomly chosen jobs are swapped. An example of the swap operator is presented below where the jobs at position 3 and 8 are swapped:

Before swap: 4 5 7 3 2 9 6 1 8

After swap: 4 5 1 3 2 9 6 7 8

Inversion move: The inversion move is a unary operator where the elements between two random crosspoints are reversed. An example of the inversion operator is presented below where the job block (2, 3, 9, 6) is inverted:

Before inversion: 4 5 7 | 3 2 9 6 | 1 8

After inversion: 4 5 7 | 6 9 2 3 | 1 8

Insertion move: In this scheme, a randomly chosen job is inserted in another random position, different from its initial position. It is also possible to move a job block. Finally, Or [27] introduces the *OrOpt* move where *k* consecutive jobs with $k \in \{1,2,3\}$ are moved from one position to another.

Shift move: Two shift moves can be defined depending on the displacement of the affected job: backward shift (noted as SHIFT_B) and forward shift (noted as SHIFT_F) moves. In a backward shift move the current sequence configuration at position *j* is moved to position *i* with $i < j$, whereas all jobs at locations *k*, with $k = i, ..., j-1$, are shifted one position backward. An example of the backward shift move is presented below with $i = 3$ and $k = 8$:

Before backward shift: 4 5 7 3 2 9 6 1 8

After backward shift: 4 5 3 2 9 6 1 7 8

Also, an example of the forward shift move is presented below with $i = 8$ and $k = 7$:

Before forward shift: 4 5 7 3 2 9 6 1 8

After forward shift: 4 5 1 7 3 2 9 6 8

2) Neighborhood size and selection scheme: If we consider a sequence *S* of *n* jobs and we generate a neighborhood using the swap move where we interchange only job *i* and job *i + 1* in the sequence without disturbing the remaining jobs, we can generate *(n-1)* different solutions. Hence, using a swap move and considering all the possible moves, the neighborhood of *S* have a size of *n∗(n-1)*. For this purpose, we generally generate only a subset of the neighborhood noted as the *k-move* neighborhood where *k* represents the neighborhood size [26]. In this latter case, the neighborhood can be visited randomly (i.e., one among all randomly generated) or in an oriented way (i.e., the best one). Indeed, the selection scheme will choose the next solution to be selected and different schemes can be used [26], such as *Best-Improve* (the best in the neighborhood), *First-Improve* (the first solution improving current in the neighborhood), *k-Improve* (a set of *k* solutions), etc.

If we consider a maximum number of evaluations as the stopping criterion, we present below two generalizations of the selection scheme that have been extracted from published works and used in this paper.

- *Descent Algorithm*: This algorithm is also known as the Hill Climbing Algorithm. In this algorithm, as shown in Figure 2, using a move operator, we generate only one solution in the neighborhood and the $Current$ solution will be updated only if the total tardiness will be improved.

$k \leftarrow 0$
while $k \leq MAX\_EVALUATIONS$ **do**
  $Next \leftarrow move(Current)$
  if $TT(Next) < TT(Current)$ **then**
    $Current \leftarrow Next$
  end if
  $k \leftarrow k + 1$
end while

Fig. 2. Descent Algorithm

- *Neighborhood Move Algorithm*: In this algorithm, using a move operator, we generate *NEIGHBORHOOD_SIZE* solutions representing the neighborhood $\mathcal{N}(Current)$, as shown in Figure 3. Then, we choose the *Next* solution which represents the $best(\mathcal{N}(Current))$, i.e., the best solution in the neighborhood, or the first one, or a random one. We can also add some perturbation policies, such as randomly assigning the best solution to the $Current$ solution.

$k \leftarrow 0$
while $k < MAX\_EVALUATIONS$ **do**
  $Generate\ \mathcal{N}(Current)$
  $Next \leftarrow best(\mathcal{N}(Current))$
  if $TT(Next) < TT(Current)$ **then**
    $Current \leftarrow Next$
  end if
  $k \leftarrow k + NEIGHBORHOOD\_SIZE$
end while

Fig. 3. Neighborhood Move Algorithm

III. EXPERIMENTAL EVALUATION

A. Data generation

The benchmark problem set consists of 160 problem tests. The instances are combinations of *N* and *M*, where *N* = {20, 50, 80, 120} and *M* = {2, 4, 8}. The processing times are generated from a uniform [1, 99] distribution. The setup times are generated according to two distributions [1, 25] and [1, 50]. This corresponds to a ratio between setup and processing times of 25% and 50% respectively. The number of parallel machines at each stage is sampled from a uniform distribution in the range [1, 4]. The probability of skipping a stage for each job is set at 0.10 and 0.40. Finally, due dates are uniformly distributed between *P(1-T - R/2)* and *P(1-T + R/2)* [28], where *T* and *R* are the tardiness factor of jobs and the dispersion range of due dates, respectively, while *P* is a lower bound of the makespan proposed by Santos et al. [29]. The pair (*T*, *R*) has values (0.3, 0.3) and (0.6, 0.3).

 ISBN: 978-1-61208-365-0

B. Computational results and discussion

All the experiments were run on an Intel Core i7 2.8 GHz processor with 8 GB of main memory. Each instance was executed 10 times. Since the evaluation of the SDST/HFFS total tardiness problem is very time consuming, all the experiments were done with a stopping criterion set to 1000 evaluations. To evaluate the different algorithms we use the relative percentage deviation (*RPD*) as shown in Equation 1.

$$RPD = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100 \quad (1)$$

where Heu_{sol} is the best total tardiness obtained by a given algorithm and $Best_{sol}$ is the best total tardiness obtained by the whole experiment. The response variable is the minimum of the 10 executions for the considered heuristic.

The first experiments are done to compare the different heuristics and neighborhood search structures. In Table I, we compare different neighborhood search structures with the EDD, SLACK, MDD and NEH heuristics. The different neighborhood search algorithms use algorithm in Figure 3 with the neighborhood size equal to 20. Note that the choice of this size was made following other numerical experiments not reported in this paper. The considered neighborhood search structures are *swap*, *OrOpt*, *shift backward*, *shift forward*, *inversion* and *insertion* moves noted as SWAP, ORPT, SH_B, SH_F, INV and INS, respectively in Table I. The results are grouped by *n* and *m* and the best results are shaded

TABLE I. COMPARISON OF THE BEST RESULT AVERAGES OF DISPATCHING RULES, CONSTRUCTIVE HEURISTICS AND NSA

Instances	EDD	SLACK	MDD	NEHT	SWAP	ORPT	SH_B	SH_F	INV	INS
20×2	52.75	142.17	97.81	105.25	**2.59**	2.71	6.71	4.58	10.36	3.14
20×8	36.62	107.33	66.69	89.10	2.03	1.82	4.04	**1.55**	7.91	1.97
50×2	56.01	108.27	80.21	88.55	**3.50**	6.12	8.23	10.13	15.02	6.34
50×4	48.00	108.24	78.29	88.27	**4.03**	4.52	9.07	5.59	15.85	4.63
50×8	40.44	83.31	65.11	73.49	4.04	3.70	8.92	**3.42**	15.91	3.47
80×2	41.41	66.93	62.48	66.35	**4.35**	6.51	10.52	8.49	16.49	8.11
80×4	44.94	69.89	56.85	61.97	**5.34**	6.27	11.13	8.29	17.76	6.65
80×8	29.62	51.59	36.31	41.93	**3.12**	4.36	7.48	4.48	11.46	4.37
120×2	47.77	66.79	58.47	61.45	**7.16**	9.07	13.84	12.60	22.39	11.38
120×4	27.77	47.82	40.35	44.63	**3.90**	5.75	9.16	6.49	11.84	7.57
120×8	47.03	63.38	59.51	58.53	7.69	**7.51**	13.93	9.51	24.04	9.54
Average	**42.94**	**83.25**	**63.83**	**70.87**	**4.34**	**5.30**	**9.37**	**6.83**	**15.37**	**6.11**

Table I show us that the *swap* neighborhood provides the best results among the tested algorithms, with an average *RPD* value of 4.34%, and the best *RPD* average for all the group instances except for the 20×8, 50×8 and the 120×8. The worst performing algorithms are the dispatching rule (EDD, SLACK, MDD) and the NEH algorithm with *RDP* averages of 42.94%, 83.25% 63.83% 70.87%, respectively. As the problem is very complex, it was expected that these latter algorithms give similar results.

After comparing the neighborhood structure, we evaluate different neighborhood search algorithm strategies. We present here three different strategies:

- The first strategy, noted *S1*, is that defined by algorithm in Figure 3 and its results are presented above in Table I.
- The second strategy, noted *S2*, is defined by a modification at the selection scheme in Figure 3. Indeed, the neighborhood can be generate from the best solution in the last neighborhood or from the best solution found by the algorithm. The choice of the solution is made randomly, i.e., by a fair coin toss.
- The third strategy, noted *S3*, evaluates the algorithm in Figure 2 which, as a reminder, represents a *hill climbing* algorithm.

Furthermore, we can improve the neighborhood search using simultaneous different *moves* [30]. Indeed such approaches can diversify the solution search space. For this purpose we retain the *swap* and the *OrOpt moves*. Thereby, we obtain 9 different algorithms whose results are presented in Table II. Under each strategy column there are three subcolumns : SWAP, ORPT and S/O representing the *swap moves*, the *OrOpt moves* and the algorithm which chooses the move randomly at each time by a fair coin toss. The results are also grouped by *n* and *m* and the best results are also shaded.

TABLE II. COMPARISON OF THE BEST RESULT AVERAGES OF DIFFERENT NEIGHBORHOOD SEARCH ALGORITHM STRATEGIES.

	S1 Strategy			*S2* Strategy			*S3* Strategy		
Instances	SWAP	ORPT	S/O	SWAP	ORPT	S/O	SWAP	ORPT	S/O
20×2	2.59	2.71	2.45	4.09	3.66	5.20	2.85	4.36	**1.39**
20×8	2.03	1.82	6.31	2.73	5.97	1.97	2.02	1.86	**1.29**
50×2	3.50	6.12	4.83	2.73	4.35	4.58	**0.96**	3.36	1.96
50×4	4.03	4.52	4.10	**0.92**	3.80	1.52	1.90	2.07	2.42
50×8	4.04	3.70	6.73	4.13	11.10	5.33	1.36	2.33	**1.12**
80×2	4.35	6.51	6.00	6.93	5.44	6.57	**0.85**	2.60	1.25
80×4	5.34	6.27	4.82	4.00	2.78	3.24	**1.03**	1.76	1.05
80×8	3.12	4.36	3.64	2.86	3.17	3.60	**1.01**	1.43	1.29
120×2	7.16	9.07	5.14	7.72	7.06	4.56	1.72	2.19	**0.46**
120×4	3.90	5.75	5.93	4.54	5.63	6.35	**0.79**	1.87	1.19
120×8	7.69	7.51	5.95	7.18	7.06	6.68	1.50	1.05	**0.97**
Average	**4.34**	**5.30**	**5.08**	**4.35**	**5.46**	**4.51**	**1.45**	**2.26**	**1.31**

The first observation is that the *S3* strategy obtains the best average with 1.45%, 2.26%, 1.31% for the SWAP, ORPT and S/O, respectively. Moreover, the *S3* strategy SWAP and S/O algorithms provide 10 of the 11 best group instance averages. Indeed, the S/O algorithm averages are better than the ORPT ones, but not as good as the SWAP ones. In fact, the results are very similar. Thus, neither changing the replacement scheme or using two different neighborhoods have made effective improvements, in general. With the *S1* strategy, S/O improves on SWAP and ORPT in only 4 and 8 instances, respectively, and with the S2 strategy, S/O improves on the other two algorithms in only 5 instances. Furthermore, the SWAP, ORPT and S/O results are better with S2 than with S1 in 5, 8 and 7 instances respectively

We now analyze the *S3* strategy results. The first interesting conclusion is that the *hill climbing* algorithm significantly improves most of the results, outperforming the other two strategies in 9, 9 and 10 instances, for SWAP, ORPT and S/O, respectively. We believe that the *S1* and *S2* strategies cannot perform better because the 1000-evaluation stopping criterion is not sufficient to reach good sequences. Moreover, with S3, the S/O algorithm improves on SWAP and ORPT in 5 and 10 instances, respectively. We can conclude that using these two neighborhoods together enhances the search results with

 ISBN: 978-1-61208-365-0

the *S3* strategy. Also, as we can see in Figure 4, both *S3-S/O* and *S3-SWAP* perform better than the other S/O algorithms strategies, in general and especially for larger instances.

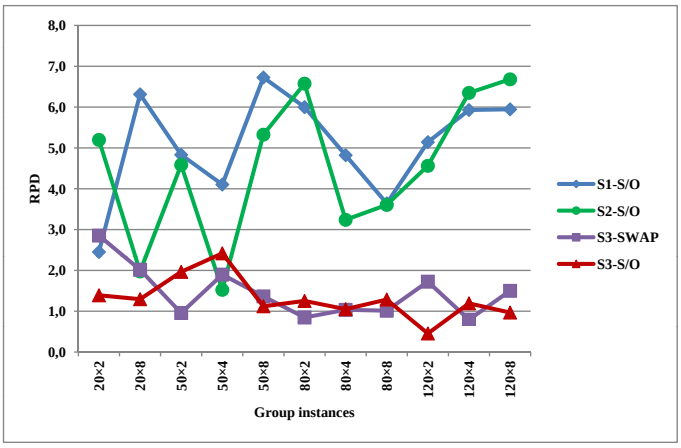

Fig. 4. Comparison of different neighborhood search algorithm strategies RPD.

It is also necessary to analyze the efficiency of the different algorithms. We measured the time in seconds that a given method needs in order to provide a solution. We remarked that all the neighborhood search algorithms have very similar CPU times, i.e., less than 1 second of deviation. Therefore, we report in Table III only the S/O algorithm with *S3* strategy, noted NSA. We also summarize there the CPU times of EDD, SLACK, MDD and NEHT grouped by n and m.

TABLE III. AVERAGE CPU TIMES OF THE DISPATCHING RULES, CONSTRUCTIVE HEURISTICS AND NSA.

Instances	EDD	SLACK	MDD	NEHT	NSA
20*2	0.00	0.00	0.07	0.07	0.36
20*8	0.01	0.01	0.12	0.81	4.19
50*2	0.00	0.01	0.18	3.90	3.31
50*4	0.01	0.02	0.20	13.75	9.54
50*8	0.04	0.04	0.35	44.58	34.68
80*2	0.02	0.03	0.25	32.34	9.94
80*4	0.05	0.04	0.28	113.09	36.41
80*8	0.17	0.17	0.45	470.76	130.81
120*2	0.03	0.04	0.26	190.43	29.62
120*4	0.12	0.11	0.41	681.66	97.34
120*8	0.49	0.45	1.24	1490.41	362.27
Average	**0.09**	**0.08**	**0.35**	**276.53**	**65.32**

The first observation is that the NEHT, executing $n * (n-1)/2$ evaluations, is very time consuming. Furthermore, concerning the NSA algorithms, we remark that dealing with the 8 stage instances notably increase the CPU time due to the evaluation of the SDST/HFFS total tardiness problem.

IV. CONCLUSION

In this work, we have introduced several algorithms to solve the hybrid flexible flowshop problem with sequence dependent setup times, which minimizes the total tardiness. We remind readers that this problem has never been addressed. We compare the behavior of different approaches as dispatching rules, constructive heuristics and a neighborhood search on the new introduced benchmarks. The results support our claim that stage skipping and setup times need to be specifically considered in the solution methods if high performance is desired, considering that this problem is very time consuming.

A perspective of this work is to enhance proposed approaches, such as parallelizing them, and to introduce a more robust algorithm such as metaheuristics.

REFERENCES

[1] M. Pinedo, Scheduling Theory: Algorithm and Systems. Prentice-Hall, 2002.

[2] R. Ruiz and C. Maroto, "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility," European Journal of Operational Research, vol. 169, no. 3, March 2006, pp. 781–800.

[3] A. Allahverdi and H. Soroush, "The significance of reducing setup times/setup costs," European Journal of Operational Research, vol. 187, no. 3, 2008, pp. 978 – 984.

[4] R. Dudek, M. Smith, and S. Panwalkar, "Use of a case study in sequencing/scheduling research," Omega, vol. 2, no. 2, 1974, pp. 253–261.

[5] G. Conner, "10 questions," Manufacturing Engineering Magazine, 2009, pp. 93–99.

[6] A. Allahverdi, C. Ng, T. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," European Journal of Operational Research, vol. 187, no. 3, 2008, pp. 985 – 1032.

[7] X. Zhu and W. E. Wilhelm, "Scheduling and lot sizing with sequence-dependent setup: A literature review," IIE Transactions, vol. 38, no. 11, 2006, pp. 987–1007.

[8] A. Vignier, J. C. Billaut, and C. Proust, "Scheduling problems of hybrid flowshop type : state of the art," RAIRO - Operations Research, vol. 33, no. 2, 1999, pp. 117–183.

[9] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. G. H. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," Annals of Discrete Mathematics, vol. 5, 1979, pp. 287–326.

[10] J. Du and J. Y. T. Leung, "Minimizing total tardiness on one machine is np-hard," Mathematics and Operations Researchs, vol. 15, 1990, pp. 438–495.

[11] R. Ruiz, F. S. Serifoglu, and T. Urlings, "Modeling realistic hybrid flexible flowshop scheduling problems," Computers and Operations Research, vol. 35, no. 4, 2008, pp. 1151 – 1175.

[12] Z. H. Jin, K. Ohno, T. Ito, and S. E. Elmaghraby, "Scheduling hybrid flowshops in printed circuit board assembly lines," Production and Operations Management, vol. 11, no. 2, 2002, pp. 216–230.

[13] M. Zandieh, S. Fatemi Ghomi, and S. Moattar Husseini, "An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times," Applied Mathematics and Computation, vol. 180, no. 1, 2006, pp. 111 – 127.

[14] H. Mirsanei, M. Zandieh, M. Moayed, and M. Khabbazi, "A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times," Journal of Intelligent Manufacturing, vol. 22, 2011, pp. 965–978.

[15] B. Naderi, R. Ruiz, and M. Zandieh, "Algorithms for a realistic variant of flowshop scheduling," Computers and Operations Research, vol. 37, no. 2, Feb. 2010, pp. 236–246.

[16] P. Gomez-Gasquet, C. Andres, and F. Lario, "An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan," Expert Systems with Applications, vol. 39, no. 9, 2012, pp. 8095 – 8107.

[17] A. Sioud, M. Gravel, and C. Gagne, "A genetic algorithm for solving a hybrid flexible flowshop with sequence dependent setup times," in Evolutionary Computation (CEC), 2013 IEEE Congress on, June 2013, pp. 2512–2516.

[18] B. Naderi, M. Zandieh, and V. Roshanaei, "Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness," The International Journal of Advanced Manufacturing Technology, vol. 41, 2009, pp. 1186–1198.

 ISBN: 978-1-61208-365-0

[19] F. Jabbarizadeh, M. Zandieh, and D. Talebi, "Hybrid flexible flowshops with sequence-dependent setup times and machine availability constraints," Computers and Industrial Engineering, vol. 57, no. 3, Oct. 2009, pp. 949–957.

[20] N. Javadian, P. Fattahi, M. Farahmand-Mehr, M. Amiri-Aref, and M. Kazemi, "An immune algorithm for hybrid flow shop scheduling problem with time lags and sequence-dependent setup times," The International Journal of Advanced Manufacturing Technology, vol. 63, 2012, pp. 337–348.

[21] R. Ruiz and T. Stutzle, "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," European Journal of Operational Research, vol. 187, no. 3, 2008, pp. 1143 – 1159.

[22] G. Mainieri and D. Ronconi, "New heuristics for total tardiness minimization in a flexible flowshop," Optimization Letters, vol. 7, no. 4, 2013, pp. 665–684.

[23] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," Omega, vol. 11, no. 1, 1983, pp. 91 – 95.

[24] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, and F. Werner, "Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria," The International Journal of Advanced Manufacturing Technology, vol. 37, no. 3-4, 2008, pp. 354–370. [Online]. Available: http://dx.doi.org/10.1007/s00170-007-0977-0

[25] R. Ruiz and J. A. Vzquez-Rodrguez, "The hybrid flow shop scheduling problem," European Journal of Operational Research, vol. 205, no. 1, August 2010, pp. 1–18.

[26] M. Prandtstetter and G. R. Raidl, "An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem." European Journal of Operational Research, vol. 191, no. 3, 2008, pp. 1004–1022.

[27] I. Or, "Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking," Ph.D. dissertation, Northwestern University, Illinois, 1976.

[28] C. Potts and L. V. Wassenhove, "A decomposition algorithm for the single machine total tardiness problem," Operations Research Letters, vol. 1, no. 5, 1982, pp. 177 – 181.

[29] D. Santos, J. Hunsucker, and D. Deal, "Global lower bounds for flow shops with multiple processors," European Journal of Operational Research, vol. 80, no. 1, 1995, pp. 112 – 120.

[30] N. Mladenovic and P. Hansen, "Variable neighborhood search," Computers and Operations Research, vol. 24, no. 11, 1997, pp. 1097 – 1100.

 ISBN: 978-1-61208-365-0