"A heuristic algorithm based on routing decisions for the No-Wait Flexible Job-Shop scheduling problem"

Domenico Daniele Nucera*, Elisa Negri*, Luca Fumagalli *

** Department of Management, Economics and Industrial Engineering of Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan – Italy (domenico.nucera@polimi.it– corresponding author -, elisa.negri@polimi.it, luca1.fumagalli @polimi.it*

Abstract: In the field of Scheduling Optimization, the Job-Shop scheduling problem entails a high level of complexity and is often solved with the aid of metaheuristic algorithms. In its Flexible version it is possible to choose the machine which will execute a specific operation. In this scenario both the scheduling and the routing problem need to be solved when aiming to optimize the production makespan. Many optimization approaches account for the routing decision by encoding it in addition to the job ordering, to let the metaheuristic algorithm account for this decision. The No-Wait Flexible Job-Shop is a variant of the problem characterized by consecutiveness constraints among operations, which need to start directly after the previous one's completion time. These constraints can be encountered for example in the pharmaceutical and metallurgic sector. In this context Timetabling Algorithms are used to decode the solution and provide a makespan given a schedule. In this paper, an approach to exploit the Timetabling Algorithm routing capabilities is discussed, in order to investigate which decision level of the optimization procedure shall execute the routing selection. Finally, possible experimental settings to validate the most convenient routing strategy are discussed.

Keywords: "No-Wait", "Flexible Job-Shop", "Routing", "Timetabling", "Scheduling

1. Introduction

Scheduling optimization is often one of the hardest tasks in an industrial facility. Often in this domain decision-making requires support from computational tools, which can require an ad-hoc implementation according to the production system's characteristics. One of the most widely adopted performance measures for a schedule is represented by the makespan, i.e. the latest instant of completion among all jobs, and most scheduling execution problems involve its minimization.

Among scheduling problems, the Job-Shop (JS) scheduling problem involves a product required to be processed by several resources along a predefined path, and it is known to be NP-Hard (Garey, Johnson and Sethi, 1976). The Flexible Job-Shop (FJS) allows operation of a job to be processed by possibly more than one resource in the problem. Every operation is thus characterized by a subset of the available resources which are entitled to execute it.

While the JS scheduling problem entails the ordering of the jobs, in the FJS the optimization procedure also requires selecting, for each operation, which one of the allowed resources shall process it. The routing among resources becomes thus a part of the optimization procedure, involving an additional decision level to the sequencing one (Pezzella, Morganti and Ciaschetti, 2008).

According to (Brandimarte, 1993), approaches to solve the FJS scheduling problem can be categorized as:

- *Concurrent*, which are based on the idea of solving the routing and scheduling problems at the same time.
- *Hierarchical*, in which different problems are solved in separate phases. Among the most frequently adopted, a first stage is aimed at solving the routing problem, and then sorting its related sequencing problem. This decomposition is driven by the fact that once a resource selection is performed, the problem reduces to a JS.

The No-Wait constraint imposes each operation to be executed directly after the previous one. Any time interval between consecutive operations would result in a violation of the No-Wait constraint. This constraint characterizes production systems that can be found in the metallurgic (Aschauer *et al.*, 2017) and pharmaceutical (Raaymakers and Hoogeveen, 2000) sector, just to make a few examples.

A No-Wait Flexible Job-Shop (NWFJS) is thus a scheduling problem in which a Job-Shop has the No-Wait constraint between its consecutive operations, which can be executed by a resource selected from a subset. Every

operation thus possesses its own subset of resources capable of executing it.

When solving No-Wait scheduling problems, a common approach is to decompose the problem into a sequencing and a timetabling one (Macchiaroli, Mole and Riemma, 1999). The sequencing part is often solved by applying a metaheuristic algorithm to search the solution space. A Timetabling Algorithm is then applied to decode the solutions by obtaining their makespan.

However, for the NWFJS no relevant contribution determines if a concurrent or hierarchical approach would be better. A comparison of different strategies is in fact missing in the literature, as we will see in the proceedings of this work. While in fact several works treat the FJS scheduling problem, the NWFJS lacks relevant benchmarks suggesting which approach, hierarchical or concurrent, should be followed. To overcome this gap, our work will present a hybrid approach to the solution of the NWFJS scheduling problem when the processing time of the operations does not depend on the selected resource. The hybrid approach can be used to gather insights on the best approach for the NWFJS scheduling problem. The proposed approach is based on a Timetabling Algorithm capable of determining the resources path allowing the first possible entry point for a job. Then we will present an encoding to couple the Timetabling Algorithm proposed with a meta-heuristic search algorithm capable of leveraging the hybrid approach proposed in order to inspect which component of the optimization procedure shall select the routing among resources.

In Section 2 we will briefly review previous works on the topic, with the problem being formally described in Section 3. In the end Section 4 will present the proposed approach.

2. Previous works

In this section we will go through previous works on the topic. In the first part, we will cover works regarding the FJS, with a second one concentrated on the NWFJS scheduling problem.

2.1 FJS works

The FJS scheduling problem possesses a vast literature. In (Xie *et al.*, 2019) several works are mentioned and discussed. In the following, we will consider some selected works according to the relevance for our investigation. (Brandimarte, 1993) presented a hierarchical approach based on a two-stage Tabu Search, exploiting the disjunctive graph representation to let neighbourhood functions act on operations on the critical path. (Mastrolilli and Gambardella, 2000) presented concurrent neighbourhood functions still leveraging the disjunctive graph representation, and then presented a Tabu Search algorithm exploiting them. Subsequent works concentrated on the problem encoding. (Pezzella, Morganti and Ciaschetti, 2008) proposed a hierarchical approach, based on a Genetic Algorithm with an encoding considering both operations sequencing and routing selection. The algorithm employs different strategies for initializing the population. (Zhang, Gao and Shi, 2011) and (Al-Hinai and ElMekkawy, 2011) proposed improved strategies for generating initial solutions, while proving their Genetic Algorithm to be better than already existing solutions. Their works showed the importance of providing to the metaheuristic search algorithm a proper initial population. (Fumagalli *et al.*, 2018) treated the accurate representation of a production system in order to solve an application case of FJS scheduling problem. In that work, attention was posed in modelling the production system and defining duties for both decision levels, the metaheuristic and the simulation one. In (Chen *et al.*, 2020) a strategy based on Reinforcement Learning is proposed to select crossover and mutation probabilities. The proposed approach resulted in a performance improvement on most instances from (Brandimarte, 1993).

2.2 NWFJS works

A limited amount of works in the literature is dedicated to the NWFJS.

The first known contribution is from (Raaymakers and Hoogeveen, 2000), in which a production system allowing overlapping operations is considered. In this work the processing time of each operation is independent of the chosen machine, and the neighbourhood function performs resources assignment taking into account their load. A Simulated Annealing algorithm is used to search the solution space, and its performance is tested against several dispatching rules on a benchmark based on an industrial case.

(Sundar, Suganthan and Chua, 2013) employs an Artificial Bee Colony to search the solution space of NWFJS in which an operation's processing time depends on the assigned resource. In the work, a solution is encoded by means of job ordering and machines assignment, in a way similar to (Pezzella, Morganti and Ciaschetti, 2008). The algorithm is executed on instances derived from (Brandimarte, 1993). The resources assignment is thus determined by the meta-heuristic algorithm, which gives preference to resources implying the minor processing time for a given operation.

In (Aschauer *et al.*, 2017), a Tabu Search is used for sequencing jobs, while resources assignment is performed by the Timetabling Algorithm. The processing time of each operation is not determined a priori of the schedule, but its minimum and maximum durations are provided as an input to the problem. In the work, the algorithm is compared against construction heuristics on instances coming from an industrial case.

(Pei *et al.*, 2020) present a Column Generation based approach for a proportionate two-sage NWFJS, in which every job is composed by two operations, involving two different production stages. Both operations of the same

job are considered to have equal duration. The adopted approach is interesting, but the limitations of the problem tackled represent a distance from multiple application cases.

2.3 Literature gaps

As it emerges from the review, the field of NWFJS scheduling optimization lacks a literature as consistent as the one of the FJS. In this context, there is a lack of evidence on which approach, concurrent or hierarchical, can lead to the best results. In our work we will thus propose an experimental strategy to investigate this aspect of NWFJS scheduling optimization. The assumption of processing time independent from the routing assigned seems reasonable from the practical point of view, since the only two presented application cases of NWFJS, i.e. (Raaymakers and Hoogeveen, 2000) and (Aschauer *et al.*, 2017) consider the processing of an operation time unrelated with the machine selection.

3. Problem description

In this work we consider a NWFJS with n jobs $J_1, \dots J_n$ and m resources $R_1, \dots, R_m$. Each job J_i shall be composed by n_op_i operations $O_{i,1}, \dots, O_{i,n\_op_i}$. Each operation O shall be characterized by a subset of resources capable of executing it. Each operation is characterized by a processing time $p_{i,j}$, being i and j respectively the indexes of the job and of the operation inside the job. All the operations of a job have to be executed respecting the predefined sequence, and no operations are allowed to overlap. Between each operation of a job the No-Wait constraint applies, meaning that each operation has to be executed after the previous one without any idle time in between. In this situation, the starting time of a job could need to be delayed for ensuring that every operation can be executed without having to wait for any resource (Pinedo, 2012). Considering $s_{i,j}$ as the starting instant of an operation $O_{i,j}$, the following condition is imposed:

$$s_{i,j+1} = s_{i,j} + p_{i,j} \quad \forall\, 1 \le i \le n, \forall\, 1 \le j < n\_op_i$$

Our objective is to minimize the total production makespan, defined as:

$$Cmax := \max\left(s_{i,n_{op_i}} + p_{i,n_{op_i}}\right) \quad \forall\, 1 \le i \le n$$

4. Experiment Proposal

In this section we will cover the design of the experimentation to inspect if and how a concurrent approach could benefit the solution of the NWFJS scheduling problem above described. In the following a Timetabling Algorithm aimed at selecting the routing of a job's operations in order to anticipate as much as possible the insertion of a job of will be presented. Then a decoding strategy capable of exploiting the Timetabling Algorithm will follow. Finally, in order to evaluate the performance of a concurrent approach with respect to the hierarchical one, possible strategies will be presented.

4.1 Adopted Timetabling Algorithm

To implement the proposed concurrent approach, it is necessary to have a Timetabling Algorithm capable of generating a routing according to the insertion necessities of a job. We are in fact interested in obtaining the optimal insertion for every job. Once a routing decision is executed, the subsequent NWJS insertion problem can be approached by inserting each job at the available instant.

Without a predefined routing already imposed, we need a Timetabling Algorithm dynamically able to select the resources in order to anticipate as much as possible the insertion point. In this way, the routing selection can be considered in function of the job insertion. We thus employ a Timetabling Algorithm which considers all the resources allowed for an operation and selects the one guaranteeing the first possible insertion for the given job, considering the previous jobs as already scheduled.

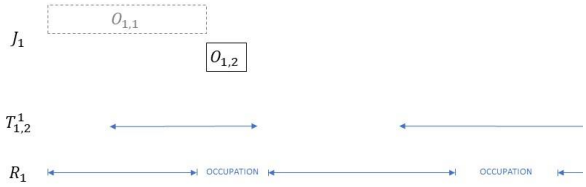

Figure 1: Time entry instants example.

Given an operation $O_{i,j}$ and a resource R_k, we consider $T^k_{i,j}$ as the set of time instants in which job J_i could start its execution guaranteeing that the operation could be executed on that machine. An example can be seen in Figure 1, in which a job is composed by two operations, and considering availabilities of resource 2 the set of time instants in which the job can start is depicted, guaranteeing that its second operation is executed on the resource. $T_{i,j}$ represents the set of time instants in which the job J_i could be executed guaranteeing proper execution of operation $O_{i,j}$ on any of the available resources. If a resource R_k is selected a priori for an operation $O_{i,j}$, then for the Timetabling Algorithm the following condition will hold:

$$T_{i,j} = T^k_{i,j}$$

Otherwise, the following expression shall be considered in order to indicate the set of time instants in which a job can be executed guaranteeing an operation's correct execution:

$$T_{i,j} = \cup_{k \in \{1,\dots,m\}} T^k_{i,j}$$

In the above expression, for a resource not entitled to execute a given operation the corresponding set of time

instants shall be an empty set. From these considerations, we determine a job's starting instant:

$$start(J_i) = \min\left(\cap_{k \in \{1,\dots,n_{op_i}\}} T_{i,j}\right)$$

This formulation allows to conceive the mechanisms required for the proposed Timetabling Algorithm to perform job insertion and routing at the same time. In case more than one resource guarantee the feasible execution of an operation at a given job insertion time instant, then the one with the smallest index is selected.

4.2 Solution Encoding

To properly represent a solution for our experiment, an encoding is needed. Two main requirements can be defined:

- The opportunity for an evolutionary algorithm to interact with the routing subproblem.
- The capability to relax for certain operations the routing imposed, thus letting the timetabling algorithm free the select a resource according to insertion necessities.

To obtain this, we start from the encoding proposed in (Pezzella, Morganti and Ciaschetti, 2008) and (Sundar, Suganthan and Chua, 2013), in which both sequencing and routing selection are represented. In particular we will refer to the latter representation, in which a sequence vector determines the job order, while every operation is characterized by the index of the resource which will account for its execution.

In this work, we are going to introduce a *concurrent index*, i.e. an encoding used to let the timetabling select the resource for a specific operation in order to anticipate the insertion point of the job it belongs. To better understand the proposed encoding, we can consider the following example, in which we denote the concurrent index as NA: we have three jobs and three resources, with the encoding for the jobs represented in Figure 2.

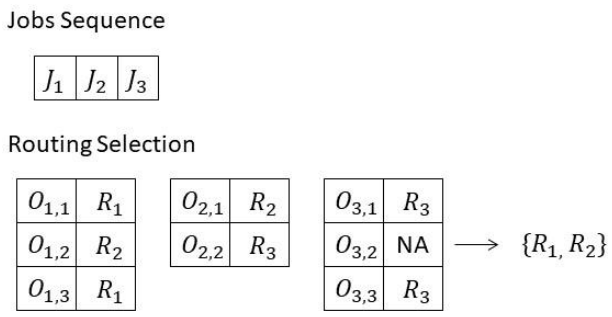

Figure 2: Example of encoding.

The three jobs are already sequenced, with a routing selection assigned for every operation but $O_{3,2}$, which in our example can be executed on either resource R_1or R_2. When decoding this solution to obtain a makespan, the timetabling algorithm will select among the two resources the one guaranteeing the first insertion point for the third job. The depiction of the Gantt chart resulting from the decoding can be seen in Figure 3, with both possible routings depicted.

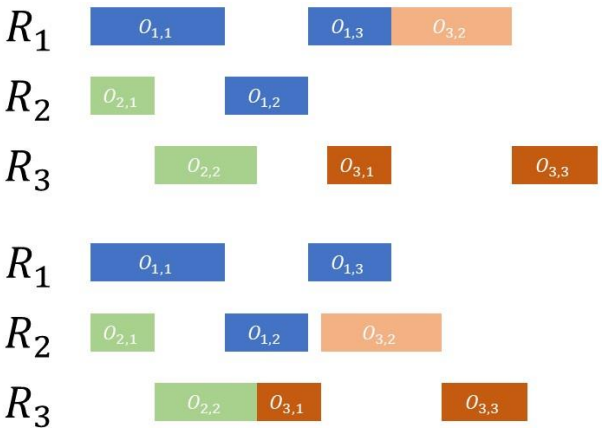

Figure 3: Gantt chart for the decoding of the example.

The timetabling algorithm will evaluate the use of both allowed resources, resorting to the use of R_2 to finally anticipate the entry point of the job. Notice that once the third job is inserted using resource R_2, in case subsequent jobs would have been present in the encoding, they would have had to consider the occupation of resources resulting from its allocation. Also, it would have been possible to allow for more than one operation in a single job to have the *concurrent index*, providing more choices to the timetabling algorithm.

4.3 Performance Evaluation

To evaluate if letting the Timetabling Algorithm route a portion of the operations can be beneficial, it can be convenient to define a *concurrent operator*, i.e. a variation of commonly used mutation operators adopted in most solutions. Certain approaches evolve individuals by randomly selecting a resource for a specific operation. This assignment can be influenced by factors like machine loads or processing times. When such a random choice has to be made, we introduce a concurrent operator with probability p_c, meaning that with probability p_c the resource selection result will be the concurrent index, thus letting the Timetabling Algorithm with the freedom to select the resource for the given operation. Given the concurrent operator, varying levels of p_c can be used to inspect the effectiveness of the concurrent strategy based on routing selection provided by the Timetabling Algorithm.

In the following, we will present possible strategies to evaluate the impact of the concurrent operator, and thus of the Timetabling Algorithm presented:

- Initialization Evaluation: certain choices in the initial population generation can benefit the performance of a search algorithm (Al-Hinai and ElMekkawy, 2011; Zhang, Gao and Shi, 2011). The proposed Timetabling Algorithm can be used to randomly generate initial populations by applying the concurrent operator. In the original

encoding it is then possible to replace the concurrent index with the resource selected by the Timetabling Algorithm. The initial population thus generated can be provided to a subsequent metaheuristic algorithm, whose performance could be used as an indicator of the usefulness of such approach.

- Survival Evaluation: an evolutionary algorithm like a Genetic Algorithm employs selection operators to preserve the fittest individuals of a population and use them to continue the search. While applying the concurrent operator, it would be possible to monitor the survival rate of individuals characterized by the presence of the concurrent index. Survival of such individuals would suggest the benefit of letting the routing selection being handled by the Timetabling Algorithm.
- Performance Evaluation: one of the simpler and clearer approaches to evaluate the benefit of the proposed concurrent approach would be to run a meta-heuristic search algorithm with varying values of p_c. A value of zero for the above mentioned parameter would imply a fully-hierarchical approach for the solution of the NWFJS scheduling problem. Varying values of p_c in different executions of the search algorithm can suggest if the proposed concurrent approach can be useful for the solution of the NWFJS scheduling problem.
- Reinforcement Learning Driven Evaluation: the approach mentioned above could be extended by adopting a Reinforcement Learning algorithm for the selection of parameters characterizing the search algorithm, among which the concurrent operator probability p_c. This strategy was used in (Chen *et al.*, 2020), and the power of the policy learning of the Reinforcement Learning algorithm could be exploited to inspect if values of p_c bigger than 0 can benefit the search procedure.

5. Conclusion

We have seen a hybrid approach to investigate if a concurrent or hierarchical solution methodology would benefit more the NWFJS scheduling problem. This results in a methodology that can be used in multiple scenarios characterized by No-Wait constraint to evaluate the most performing routing approach. The hybrid approach proposed could be considered also a solution strategy in itself. While in fact a representation accounting for routing decisions is necessary for the optimization procedure, a search algorithm could benefit from an integration with the heuristic of letting the Timetabling Algorithm procedure select resources to anticipate the insertion of a job. While there is a gap between research on the FJS scheduling problem and its counterpart characterized by the No-Wait constraint, advancements in techniques like Optimal Job Insertion (Bürgy and Gröflin, 2017) and Reinforcement Learning (Chen *et al.*, 2020) can possibly be exploited to reduce the gap in solving such scheduling problem. The introduction of IT technologies at the shop floor, leading to the paradigm of the Cyber-Physical System, can possibly lead to the inclusion of real-time information coming from the resources in the scheduling framework (Negri *et al.*, 2020; Ragazzini *et al.*, 2020). In this paradigm, insights coming from the shop floor could be integrated in the scheduling process, leading to new heuristics and hybrid algorithms for optimization. Possible future works could regard the implementation of a tool to produce computational experiments, in order to apply the proposed methodology in the evaluation of possible routing strategies for NWFJS scheduling scenarios.

References

Al-Hinai, N. and ElMekkawy, T. Y. (2011) "An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem," *Flexible Services and Manufacturing Journal*, 23(1), pp. 64–85.

Aschauer, A. *et al.* (2017) "An efficient algorithm for scheduling a flexible job shop with blocking and no-wait constraints," *IFAC-PapersOnLine*, 50(1), pp. 12490–12495.

Brandimarte, P. (1993) "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations research*, 41(3), pp. 157–183.

Bürgy, R. and Gröflin, H. (2017) "The no-wait job shop with regular objective: a method based on optimal job insertion," *Journal of Combinatorial Optimization*, 33(3), pp. 977–1010.

Chen, R. *et al.* (2020) "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem," *Computers & Industrial Engineering*, 149, p. 106778.

Fumagalli, L. *et al.* (2018) "A novel scheduling framework: Integrating genetic algorithms and discrete event simulation," *International Journal of Management and Decision Making*, 17(4), pp. 371–395.

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

Macchiaroli, R., Mole, S. and Riemma, S. (1999) "Modelling and optimization of industrial manufacturing processes subject to no-wait constraints," *International Journal of Production Research*, 37(11), pp. 2585–2607.

Mastrolilli, M. and Gambardella, L. M. (2000) "Effective neighbourhood functions for the flexible job shop problem," *Journal of scheduling*, 3(1), pp. 3–20.

Negri, E. *et al.* (2020) "Field-synchronized Digital Twin framework for production scheduling with uncertainty," *Journal of Intelligent Manufacturing*, pp. 1–22.

Pei, Z. *et al.* (2020) "A column generation-based approach for proportionate flexible two-stage no-wait job shop

scheduling," *International Journal of Production Research*, 58(2), pp. 487–508.

Pezzella, F., Morganti, G. and Ciaschetti, G. (2008) "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, 35(10), pp. 3202–3212.

Pinedo, M. (2012) *Scheduling*. Second Edition. Springer.

Raaymakers, W. H. M. and Hoogeveen, J. A. (2000) "Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing," *European Journal of Operational Research*, 126(1), pp. 131–151.

Ragazzini, L. *et al.* (2020) "Tolerance Scheduling for CPS," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*. IEEE, pp. 487–492.

Sundar, S., Suganthan, P. N. and Chua, T. J. (2013) "A swarm intelligence approach to flexible job-shop scheduling problem with no-wait constraint in remanufacturing," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, pp. 593–602.

Xie, J. *et al.* (2019) "Review on flexible job shop scheduling," *IET Collaborative Intelligent Manufacturing*, 1(3), pp. 67–77.

Zhang, G., Gao, L. and Shi, Y. (2011) "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Systems with Applications*, 38(4), pp. 3563–3573.