A Relax-and-Solve Algorithm for the Ordered Flow-Shop Scheduling Problem

Mostafa Khatami*, Amir Salehipour†
School of Mathematical and Physical Sciences, University of Technology Sydney, Australia,
Email: *mostafa.khatami@student.uts.edu.au, †amir.salehipour@uts.edu.au

***Abstract*—In the ordered flow-shop scheduling problem the processing times follow specific structures. We propose a relax-and-solve matheuristic for the ordered flow-shop problem to minimize the makespan, which is proven to be NP-hard. We compare the performance of our method and that of the state-of-the-art methods, and show that the proposed method is capable of reporting new best solutions for a large number of instances, and has the average gap of as low as 0.046% from the best known solutions.**

Keywords: scheduling, flow-shop, matheuristic

I. Introduction

As a subcategory of the flow-shop scheduling problem, the ordered flow-shop deals with the case where there are structured properties for the processing times, in particular, jobs and machines are described by the following two conditions: (1) if the processing time of a job j is smaller than that of another job k on some machine, then job j has smaller processing time than job k on all machines, and (2) if the processing time of a job on a machine r is smaller than that on another machine q, then all jobs have smaller processing time on machine r than on machine q. Those structured properties represent many real-world industrial procedures, where the processing time of the jobs are related to the physical characteristics of the jobs and/or machines [1]. Typically, the processing of a small-sized job is more time-consuming on any machine, or a machine with an old technology is slow in processing any job. Two real-world examples include the manufacturing facility of liquid crystal display panels and the painting process of expensive wooden doors [2].

The problem of minimizing the makespan on the ordered flow-shop problem was first introduced by [3]. For the problem when the largest processing times occur on the first (last) machine, [1] proved that the longest processing time (LPT) (shortest processing time (SPT)) first dispatching rule is optimal. The problem becomes NP-hard when the largest processing times do not occur on either the first or the last machine [4]. An intriguing result was obtained by [5], where they showed that a pyramidal-shaped sequence is optimal for the ordered flow-shop problem. A pyramidal-shaped sequence consists of two sub-sequences, where in the first sub-sequence the jobs are ordered by the SPT rule, and in the second sub-sequence the jobs are sequenced in the LPT order. Recently, [6] utilized the pyramidal-shaped property and proposed two heuristics and an iterated local search (ILS) algorithm for the ordered flow-shop problem. For a detailed review of earlier studies, we refer the interested reader to [4]. [6] provided comprehensive computational comparisons of the state-of-the-art methods.

The ILS algorithm of [6] is the only published meta-heuristic for the ordered flow-shop problem with the makespan minimization. The main contribution of this study is to propose a new algorithm for the problem that uses an exact solver within a heuristic framework, and for that purpose we propose the relax-and-solve (R&S) matheuristic. By solving a set of 120 challenging instances we show that the new algorithm outperforms standard optimization solvers, and has a good performance in comparison to the state-of-the-art methods. The remainder of this paper is organized as follows. We define the problem in Section II and present the solution method in Section III. In Section IV, we report the results of our computational experiments, and conclusions and future research directions are provided in Section V.

II. Problem definition and formulation

Given the sets $J = \{1, \ldots, n\}$ of jobs and $M = \{1, \ldots, m\}$ of machines, all jobs must be processed on all machines in the order 1 to m. The processing time of job $j \in J$ on machine $r \in M$ is denoted by p_{rj}, and the processing times meet the followings:

1) if for any two jobs $j, k \in J, p_{rj} < p_{rk}, r \in M$, then $p_{qj} \leq p_{qk}, \forall q \in M$; and,
2) if for any two machines $r, q \in M, p_{rk} < p_{qk}, k \in J$, then $p_{rj} \leq p_{qj}, \forall j \in J$.

We consider the problem with the permutation assumption, i.e., the job processing orders on all m machines are identical. Additionally, all jobs are available at the time zero and preemption of jobs is not allowed, meaning that once the execution of a job is started on some machine, it cannot be interrupted by other jobs. Also, each machine can perform at most one job at a time. We consider the objective of minimizing the length of the schedule, i.e., makespan denoted by C_{max}.

Among the mixed-integer programming (MIP) formulations for the flow-shop scheduling problem, we implement the formulation proposed by [7], denoted by Model I in the following, for the ordered flow-shop problem. According to [8], Model I is the best performing model for solving the permutation flow-shop problem. In Model I, binary decision variables z_{ji} take the value of 1 if job j is assigned to position $i \in J$ in the sequence, and 0 otherwise. In addition, the non-negative decision variables x_{ri} and y_{ri} are used to indicate the idle time of machine r before starting the job in position i, and the idle time of the job in position i after finishing its process on machine r, respectively.

Model I

$$z = \min C_{max} = \min(\sum_{j\in J} p_{mj} + \sum_{i\in J} x_{mi}) \quad (1)$$

subject to

$$\sum_{j\in J} z_{ji} = 1, \quad i \in J, \quad (2)$$

$$\sum_{i\in J} z_{ji} = 1, \quad j \in J, \quad (3)$$

$$\sum_{j\in J} p_{rj} z_{j1} + x_{r1} + y_{r1} = x_{r+1,1}, \quad r \in M \setminus \{m\}, \quad (4)$$

$$\sum_{j\in J} p_{rj} z_{j,i+1} + x_{r,i+1} + y_{r,i+1} = \sum_{j\in J} p_{r+1,j} z_{ji} + x_{r+1,i+1} + y_{ri}, \quad i \in J \setminus \{n\}, r \in M \setminus \{m\}, \quad (5)$$

$$z_{ji} \in \{0,1\}, \quad i,j \in J, \quad (6)$$

$$x_{ri} \geq 0, \quad y_{ri} \geq 0, \quad r \in M, i \in J. \quad (7)$$

The objective function (eq. (1)) minimizes the makespan. The assignment constraints (2) and (3) ensure that each position is filled with only one job and exactly one position is assigned to each job. The job-adjacency and the machine-linkage constraints presented in (4) and (5) ensure that the process of job in position i cannot be started on machine $r+1$ until its process on machine r is finished, and the process of job in position $i+1$ cannot be started on machine r, until the process of the job in position i on that same machine is finished. Constraints (6) and (7) ensure $z_{ji} \in \{0,1\}$, $x_{ri} \geq 0$ and $y_{ri} \geq 0$.

III. The proposed relax-and-solve method

In this section we propose an efficient relax-and-solve (R&S) heuristic algorithm for the ordered flow-shop scheduling problem. The R&S algorithm was designed by [9, 10, 11, 12]. Given an initial sequence, the R&S algorithm obtains an improved solution through iterative relaxation of a subset (of the given sequence) of operations, and solving the subset, i.e., re-ordering and re-scheduling by using optimization techniques. The "relax" phase of the R&S allows those operations in the subset to be able to change their execution order, whereas the "solve" phase ensures the subset is re-ordered and a feasible schedule is constructed. Algorithm 1 summarizes the R&S heuristic algorithm.

Algorithm 1: The R&S heuristic algorithm.

Input: The initial sequence π of performing the operations on the machines, parameter K.
$k := 1$;
while *the stopping condition is not met and* $k \leq K$ **do**
  Apply neighborhood k (N_k) to relax $\pi' \subset \pi$;
  Solve the problem by using an optimization solver;
end
return *The best obtained schedule (the solution);*

The pyramidal-shaped property of the ordered flow-shop problem significantly reduces the size of the set of all candidate feasible solutions, more precisely from $n!$ to 2^{n-1} as discussed in [6]. Therefore, we utilize this property in the development of the neighborhoods for the proposed R&S algorithm. In what follows we discuss the components of the proposed R&S heuristic.

A. Solution representation

Since the problem under study is considered with the permutation assumption, any sequence of executing the jobs on the machine is then feasible. We therefore present a feasible solution by a sequence of jobs in positions 1 to n, i.e., a permutation, where job $J_{(i)}$ represents the job in position i in the sequence; see Figure 1.

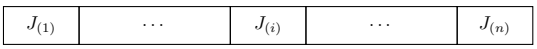

Fig. 1. The solution representation in the proposed R&S algorithm.

B. Initial solution

We utilize the pair-insert heuristic algorithm of [6] to generate an initial solution for the R&S algorithm. The pair-insert is a three-step procedure that utilizes the pyramidal-shaped property of the problem, as well as the general idea of the well-known NEH algorithm of [13]. Step 1 sorts the jobs in non-increasing order of their total processing times. Step 2 generates a two-job partial sequence and Step 3 completes the partial sequence by adding unassigned jobs in pairs. We refer the interested reader to [6] for details on the pair-insert algorithm.

C. Neighborhoods

The main body of the R&S algorithm, which is presented in Algorithm 1, applies the relax operation by utilizing a set of neighborhoods. We present three neighborhoods, each of which applies a unique relaxation method that results in two relaxed subsets (of performing operations). Let π denote the complete sequence. From π, two subsets $\pi'_1, \pi'_2 \subset \pi$ are selected to be relaxed, and that by letting variables $z_{ji} \in \{0,1\}, \forall i \in \pi'_1 \cup \pi'_2$ (see Model I), i.e., letting Model I decide on the values of z_{ji}. For the remaining variables, i.e., $z_{ji}, \forall i \notin \pi'_1 \cup \pi'_2$, their values are kept as they appear in π. In other words, those variables are treated as parameters. The solve operation consists of solving Model I by using an optimization solver. It is clear that Model I includes only a small number of binary variables, and can therefore be solved by significantly less efforts.

Selection of the relaxed subsets π'_1 and π'_2 results in new "smaller" optimization problems that have less variables and constraints than Model I associated with the original problem. Also, changing the relaxed subsets results in different optimization problems. An optimization problem generated in a neighborhood is hereafter called a "sub-problem" of that neighborhood. For example, given $\pi = (J_{(1)}, \ldots, J_{(i')}, J_{(i'+1)}, \ldots, J_{(i'')}, J_{(i''+1)}, \ldots, J_{(n)})$, we generate a sub-problem where the decision variables related to $J_{(1)}$ to $J_{(i')}$ and $J_{(i''+1)}$ to $J_{(n)}$ (i.e., z_{j1} to $z_{ji'}$

and $z_{j,i''+1}$ to z_{jn} in Model I) are optimized. We treat the decision variables related to $J_{(i'+1)}$ to $J_{(i'')}$, i.e., $z_{j,i'+1}$ to $z_{ji''}$ as parameters.

We relax two equally-sized subsets π'_1 and π'_2 in each neighborhood, due to the pyramidal-shaped property. We denote the neighborhoods by N_1, N_2 and N_3, and the size of the subsets by n_1, n_2 and n_3, associated with N_1, N_2 and N_3, where, e.g., n_1 denotes the number of jobs that are relaxed in π'_1 (and also in π'_2) in N_1. A certain number of sub-problems is solved in each neighborhood, while ensuring that any part of the sequence is subject to optimization at least once within each neighborhood. The number of sub-problems that are solved in each neighborhood N_1, N_2 and N_3 is equal to $\lfloor \frac{n}{n_1} \rfloor$, $\lfloor \frac{n}{n_2} \rfloor$ and $\lfloor \frac{n}{n_3} \rfloor$. Next, we explain the mechanism of selecting π'_1 and π'_2 for each neighborhood.

1) Neighborhood N_1: The pyramidal-shaped property of the problem is considered in the mechanism of selecting π'_1 an π'_2 in N_1. Let $S_1 = \{1, \ldots, \lfloor \frac{n}{n_1} \rfloor\}$ be the set of all sub-problems generated by N_1. Then, π'_1 an π'_2 in the sub-problem $s \in S_1$ are the sth and the $(\lfloor \frac{n}{n_1} \rfloor - s + 1)$th parts of the sequence, respectively. We note that each part of the sequence contains n_1 jobs. As an example, for $s = 1$, π'_1 an π'_2 are the first and the last n_1 jobs, respectively, that means π'_1 includes jobs 1 to n_1 and π'_2 includes jobs $n - n_1 + 1$ to n, as shown in Figure 2. The reason behind selecting N_1 as the first neighborhood in the algorithm is that the initial solution of the algorithm, that is obtained by the pair-insert heuristic, is a pyramidal-shaped sequence.

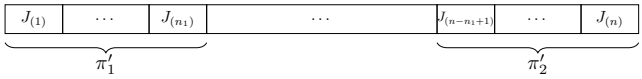

Fig. 2. Selection of π'_1 and π'_2 in the first sub-problem of N_1.

2) Neighborhood N_2: The mechanism of selecting π'_1 an π'_2 in N_2 is similar to the rolling horizon method. Let $S_2 = \{1, \ldots, \lfloor \frac{n}{n_2} \rfloor\}$ be the set of all sub-problems generated by N_2. Then, π'_1 an π'_2 in the sub-problem $s \in S_2$ are the sth and the $(s+1)$th part of the sequence, respectively. Similar to N_1, each part of the sequence contains n_2 jobs. As an example, for $s = 1$, π'_1 an π'_2 are the first and the second n_2 jobs, respectively, that means π'_1 includes jobs 1 to n_2 and π'_2 includes jobs $n_2 + 1$ to $2n_2$, as presented in Figure 3.

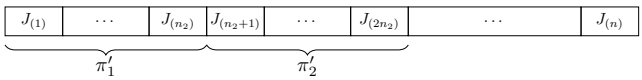

Fig. 3. Selection of π'_1 and π'_2 in the first sub-problem of N_2.

3) Neighborhood N_3: The mechanism of selecting π'_1 an π'_2 in N_3 is a disjoint version of the mechanism used in N_2. Precisely, let $S_3 = \{1, \ldots, \lfloor \frac{n}{n_3} \rfloor\}$ be the set of all sub-problems generated by N_3. Then, π'_1 and π'_2 in the sub-problem $s \in S_3$ are the sth and the $(s+2)$th part of the sequence, respectively. Like in N_1 and N_2, each part of the sequence contains n_3 jobs. As an example, for $s = 1$, π'_1 an π'_2 are the first and the third n_3 jobs, respectively, that means π'_1 includes jobs 1 to n_3 and π'_2 includes jobs $2n_3 + 1$ to $3n_3$, as depicted in Figure 4.

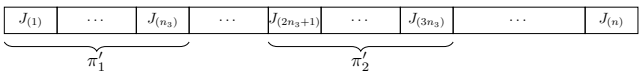

Fig. 4. Selection of π'_1 and π'_2 in the first sub-problem of N_3.

We will discuss the order of applying the neighborhoods in Section IV.

D. Stopping criterion

Two stopping criteria are considered for the algorithm. First, a time-limit criterion that works as follows is applied. Before performing each neighborhood, if the total elapsed computation time exceeds the time limit T, the algorithm stops. Otherwise, the neighborhood is completely explored. The second criterion terminates the algorithm if the algorithm goes back to a visited neighborhood with no improvement in the objective function. That is, if the best objective function is obtained in neighborhood k, and there has been no improvement with other neighborhoods, the algorithm then stops. The reason for applying this criterion is that if no improvement is gained in the neighborhoods other than k, the algorithm returns to the same neighborhood where it reached the best obtained solution. Although, because the sub-problems are heuristically solved by setting the time-limit t for the solver, finding an improving solution in the same neighborhood is unlikely.

IV. Computational experiments

In this section we report the computational results of the proposed R&S algorithm on a benchmark `T` in [6]. The benchmark consists of a set of 120 challenging instances ranging from 20 to 500 jobs, i.e., $n = \{20, 50, 100, 200, 500\}$, and 5 to 20 machines, i.e., $m = \{5, 10, 20\}$. We implement the R&S algorithm in the programming language Python version 2.7, and solve the underlying mathematical program by using the Gurobi version 8.0.0 [14]. Apart from a time limit that we use as a stopping criterion for Gurobi, we set the value of the remaining parameters of the solver to their default values. All computational experiments were performed on a standard PC with Intel® Core™ i5-7500 CPU clocked at 3.40GHz with 8GB of memory under Linux ubuntu 16.04 operating system. We compare the performance of the proposed R&S with that of ILS and CPLEX that are reported in [6].

Recall that R&S includes two time limits. The first time limit, denoted by t, is for the solver Gurobi, i.e., Gurobi is allowed to spend t seconds on solving each sub-problem. We set $t = 20$ because it leads to good quality solutions in a reasonable amount of time. We observe that smaller

or greater values of t led to poor quality solutions or unnecessarily long run time for the algorithm. The second time limit, which we denote by T, is the total time limit of R&S. Because of the different instance sizes (number of jobs and machines) we consider parameter $T = 2 \times n \times T_m$, i.e., as a function of the number of jobs and machines, where $T_m = 0.8, 1.0, 1.2$ for $m = 5, 10, 20$ is a machine-dependent coefficient. We set the relaxation size of sub-problems, i.e., parameters n_1, n_2 and n_3 to 15, 20 and 15 for N_1, N_2 and N_3. We apply the neighborhoods in order N_1, N_2, N_3 and that for two times.

Table I summarizes the performance of three solution methods of R&S, ILS and CPLEX [15]. We compare the methods across three criteria of "Best", "Gap" (from the best solution, in %), and "Time" (in seconds). The value of gap is obtained as $\frac{z-z^*}{z*} \times 100$, where z is the objective function value delivered by the solution method and z^* is the best known objective function value obtained by at least one of three solution methods.

TABLE I
SUMMARY OF THE OUTCOMES OF DIFFERENT SOLUTION METHODS.

	R&S	ILS	CPLEX
Best	51	85	39
Gap (%)	0.046	0.012	0.197
Time (seconds)	280.66	27.44	2694.25

According to Table I, the performance of R&S is superior to solving Model I with CPLEX, because R&S obtains a larger number of best solutions, and that with a smaller gap. Both R&S and ILS methods have small values of gap, which are less than 0.05%. While the proposed R&S method does not outperform ILS, it has a promising performance because R&S obtains the best solution in 51 of the instances. That corresponds to about 42% of instances. This very good performance along with the straightforward implementation of the proposed R&S method are among the main benefits of the R&S algorithm. With respect to the computational time, the R&S time is almost 280 seconds on average, that is small enough for real-world applications.

The instance size-wise analyses reported in Table II, show that the performance of R&S is reliable among different sizes of the instances, and even in large instances its gap from the best solution is less than 0.09%.

TABLE II
SUMMARY OF THE OUTCOMES OVER DIFFERENT INSTANCE SIZES.

	R&S		ILS		CPLEX	
Size	Best	Gap %	Best	Gap %	Best	Gap %
Small	38	0.031	33	0.022	39	0.029
Medium	13	0.056	42	0.002	0	0.365
Large	0	0.087	10	0.000	0	0.359

V. CONCLUSION

We investigated solving the ordered flow-shop scheduling problem with the objective of minimizing the makespan. We utilized the pyramidal-shaped property of the problem and designed an efficient relax-and-solve (R&S) algorithm that includes three neighborhood structures. We solved a set of 120 benchmark instances and compared the outcomes and that of the ILS algorithm and the solver CPLEX. We showed that our R&S obtains good quality solutions. Future researches may focus on proposing new neighborhoods for the R&S method.

ACKNOWLEDGMENTS

Mostafa Khatami is the recipient of UTS International Research Scholarship (IRS) and UTS President's Scholarship (UTSP). Amir Salehipour is the recipient of an Australian Research Council Discovery Early Career Researcher Award (project number DE170100234) funded by the Australian Government.

REFERENCES

[1] Smith, M. L., Panwalkar, S. S., and Dudek, R. A. "Flowshop sequencing problem with ordered processing time matrices". *Management Science* 21.5 (1975), 544–549.

[2] Lee, K., Zheng, F., and Pinedo, M. L. "Online scheduling of ordered flow shops". *European Journal of Operational Research* 272.1 (2019), 50–60.

[3] Smith, M. L. "A critical analysis of flow-shop sequencing". PhD thesis. Texas Tech University, 1968.

[4] Panwalkar, S., Smith, M. L., and Koulamas, C. "Review of the ordered and proportionate flow shop scheduling research". *Naval Research Logistics (NRL)* 60.1 (2013), 46–55.

[5] Smith, M. L., Panwalkar, S. S., and Dudek, R. A. "Flowshop sequencing problem with ordered processing time matrices: A general case". *Naval Research Logistics Quarterly* 23.3 (1976), 481–486.

[6] Khatami, M., Salehipour, A., and Hwang, F. "Makespan minimization for the m-machine ordered flow shop scheduling problem". *Computers & Operations Research* 111 (2019), 400–414.

[7] Wagner, H. M. "An integer linear-programming model for machine scheduling". *Naval Research Logistics Quarterly* 6 (1959), 131 – 140.

[8] Tseng, F. T., Stafford Jr., E. F., and Gupta, J. N. D. "An empirical analysis of integer programming formulations for the permutation flowshop". *Omega* 32 (2004), 285 –293.

[9] Salehipour, A. "A heuristic algorithm for the Aircraft Landing Problem". *22nd International Congress on Modelling and Simulation*. Modelling, Simulation Society of Australia, and New Zealand Inc.(MSSANZ). 2017.

[10] Salehipour, A, Ahmadian, M., and Oron, D. "Efficient and simple heuristics for the Aircraft Landing Problem". *Matheuristic 2018 International Conference*. 2018.

[11] Ahmadian, M. M., Salehipour, A., and Kovalyov, M. “An Efficient Relax-and-Solve Heuristic for Open-Shop Scheduling Problem to Minimize Total Weighted Earliness-Tardiness”. *Available at SSRN 3601396* (2020).

[12] Ahmadian, M. M., Salehipour, A., and Cheng, T. “A meta-heuristic to solve the just-in-time job-shop scheduling problem”. *European Journal of Operational Research* 288.1 (2021), 14 –29.

[13] Nawaz, M., Enscore, E. E., and Ham, I. “A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem”. *Omega* 11.1 (1983), 91 –95.

[14] Gurobi Optimization, L. *Gurobi Optimizer Reference Manual*. 2018.

[15] CPLEX, I. I. *version 12.8.0*. Armonk, New York, U.S.: IBM Corp., 2017.