The Meaning of Permutation in Reentrant Permutation Flow Shop Problems

Itamar Segal[1], Tal Grinshpoun[1][0000−0002−4106−3169], Hagai Ilani[2][0000−0003−3548−1572], and Elad Shufan[2][0000−0001−7960−5798]

[1] Ariel University, Ariel, Israel
itamar.segal@msmail.ariel.ac.il, talgr@ariel.ac.il
[2] Shamoon College of Engineering, Ashdod, Israel
hagai@sce.ac.il, elads@sce.ac.il

Abstract. In a flow shop, jobs are serially processed on a set of machines, and the machine order is the same for all the jobs. In a permutation flow shop, there is an additional assumption that the order in which jobs enter the machines is the same on each machine. While the meaning of "permutation" is clear for a flow shop, it is more ambiguous for a reentrant flow shop. In a reentrant flow shop, jobs are processed on some machines more than once. Then, there are several ways to understand the meaning of permutation. We indicate that different researchers use the term "permutation" for different assumptions. Our effort is to clear up this ambiguity. This is significant for definition clarity when studying the reentrant scenario. Moreover, the various definitions influence proposed heuristics for solving the problem and, by that, also influence the quality of the resulting solutions. We show this through basic examples as a first step towards more extensive experiments.

Keywords: Permutation Flow shop, Reentrant Flow Shop, Heuristics

1 Introduction

Scheduling in the manufacturing industry involves assigning jobs to machines and determining their order to optimize some criteria [1,22]. Many manufacturing layouts take the form of flow shops in which jobs progress from one machine to the next machine in a serial order without ever visiting the same machine twice [22]. Flow shop problems that do not allow sequence changes between machines are called **permutation flow shops** (PFS). In this class of problems, jobs are processed by a series of machines in precisely the same order [25]. A PFS problem is thus characterized by the same machine order for all jobs (flow shop) and the same job order for each machine (permutation).

In classical scheduling, it is typically assumed that each job visits any given machine at most once [1]. Contrary to that, many practical scenarios are of **reentrant shops**, in which a job may be processed by some of the machines several times [9]. In some industries, including signal processing and semiconductor wafer manufacturing, product design may call for jobs to recirculate or revisit a stage in the manufacturing process [24]. Reentrant shops can also be found in manufacturing facilities such as textile fabric manufacturing processes and mirror manufacturing systems [30]. A **reentrant flow**

shop (RFS) problem is distinguished from the classical flow shop problem by allowing jobs to be processed repeatedly at some machines.

An RFS is characterized by a reentrant pattern ϕ noting the machine order for the jobs. Explicitly, let ϕ_i be the i-th stage visited by each job, where $\phi_i \in \{1, 2, \ldots, m\}$. The reentrant pattern $\phi = (\phi_1,\phi_2,...,\phi_{|\phi|})$ is the sequence in which jobs visit the machines in the shop. The *reentrant property* occurs when at two stages in the sequence, $\phi_i = \phi_j$ for $i \neq j$. In maximal generality, the reentrant pattern determined by the manufacturing process may be arbitrary, provided all jobs follow the same sequence (i.e., flow shop) and at least one machine is revisited. Nonetheless, there are some special reentrant patterns that commonly appear in real-life manufacturing processes. One such pattern is the cyclic pattern [10], also called RFS with full loops [30]. In a cyclic-reentrant flow shop, all the jobs make l passes $(l > 1)$ through all the machines, and each pass goes through all the machines in order $1, 2, \ldots, m$. For $l = 2$ as an example, the obtained pattern is $\phi = (1, 2, \ldots, m, 1, 2, \ldots, m)$. This work assumes a cyclic pattern for the RFS. An example of cyclic RFS is in the assembly of electronic circuits stacked on top of each other [10]; each time a new circuit is connected, the same set of machines is visited. The cyclic pattern is extensively studied also because it is possible to describe any reentrant pattern by setting the processing time on some machines in some passes to be zero. Nevertheless, for reasons of clarity, we hereby consider non-zero processing times in each machine at each pass, that is, a true cyclic RFS.

An RFS problem with permutation characteristics is known as a **reentrant permutation flow shop** (RPFS) problem. The main contribution of this article regards the notion of **permutation** within the RPFS context. We claim that the use of the word "permutation" in RPFS takes on several different meanings, which naturally may create confusion. We offer a clear way to describe the different definitions of RPFS. We do not propose to abolish the various definitions but rather propose to address several permutation types. In the proposed approach, four different types of RPFS are obtained; in the literature, we found that researchers use three of them under the name RPFS.

The four permutation types are described in Section 2, together with a literature review mainly concerning RPFS with a cyclic pattern. In Section 3, we show that the different permutation types have meaning not only in terms of the clarity of representation but also from the point of view of heuristic construction. We consider Palmer's slope heuristic [20] for each of the four presented types and demonstrate it through simple examples. A discussion in Section 4 concludes the paper.

2 RPFS Permutation Types

Consider an RFS problem with a cyclic reentrant pattern. It consists of n jobs, m machines, and l levels. A level in a cyclic pattern is a pass of a job in all the machines in the order $1, 2, \ldots, m$. A level is also called a cycle or loop. A specific level of a specific job is also called a sub-job [8]. To show the ambiguity regarding the meaning of "permutation" in the RPFS literature, we present the following quotes that use contradicting definitions.

Definition 1: "Every job can be decomposed into several layers each of which starts on M_1 and finishes on M_m. In the RFS case, if the job ordering is the same on any machine at each layer, then no passing is said to be allowed, since no job is allowed to pass any former job. The RFS scheduling problem in which no passing is allowed, is called a RPFS problem." — quote taken from [6].

Definition 2: "Our considered m-machine reentrant permutation flow shop scheduling problem (MRPFSSP) can be viewed as a special kind of non reentrant flow shop scheduling problem (FSSP). If each job is decomposed into L sub-jobs and the reentrant-based precedence constraints among all sub-jobs are satisfied, MRPFSSP can be treated as an imaginary FSSP with nL sub-jobs." — quote taken from [23].

The first definition is more restrictive than the second. There is a third, intermediate definition used for general reentrant patterns.

Definition 3: "Pan and Chen (2003) developed three mixed integer models for the reentrant flow-shop problem . . . In these models, the job sequence for every machine is the same in each level and it is not allowed that higher levels preempt lower ones . . . Lee et al. (2011) relaxed the constraints of level permutation set by Pan and Chen (2003) in order to get better objective values. This relaxation makes it possible to mix job levels, i.e., jobs on higher levels can be processed on a machine k before jobs on a lower level." — quote taken from [15].

Table 1 presents a summary of the literature regarding RPFS with the cyclic pattern over the last two decades. The table shows that the definitions are used interchangeably over the years and are all referred to as RPFS. This creates an ambiguity that we aim to correct. It is noteworthy that:

- The majority of studies used Definition 1.
- Most studies that applied Definition 2 were related to the specific case of two-machine problems.
- Only a few studies used Definition 3.

A key insight that can be deduced from the above definitions, is that to properly handle the permutation characteristic in RFS problems, two issues should be considered:

1. Job passing: Is it allowed for the job order to be different at different levels?
2. Level passing: Is it allowed for level k of job j to be scheduled before level k' of another job j' where $k' < k$?

For Definition 1, the answer to both questions is negative: there is a single job order and this order is kept in all the levels. In addition, level passing is forbidden, i.e., level $k + 1$ of a job does not precede level k of a different job. We suggest to term this RPFS type as **Passing Prohibited** (PP). This term has a similar meaning to the "no passing" term used with respect to RPFS, for example, in [21,6].

For Definition 2, the answer to both questions is positive: the jobs are practically divided into sub-jobs, and the order of sub-jobs is not restricted as long as precedence constraints are kept, i.e., a level $k + 1$ of a job does not precede level k of the same job. We suggest to term this RPFS type as **Passing Allowed** (PA).

For the intermediate Definition 3, the answer to the first question is negative; a single job order is kept in all the levels. However, the answer to the second question is positive, i.e., level passing is allowed. We suggest to term this RPFS type as **Job Passing Prohibited** (JPP).

Year	**Reference**	**Permutation definition**	**Objective**	**Notes**
2003	[21]	Definition 1, PP	Makespan	
2006	[3]	Definition 1, PP	Makespan	
2007	[5]	Definition 1, PP	Makespan	
2007	[7]	Definition 2, PA	Makespan	Two-machine problem
2008	[6]	Definition 1, PP	Makespan	
2008	[17]	Definition 2, PA	Makespan	Two-machine problem
2008	[8]	Definition 2, PA	Makespan	
2009	[4]	Definition 1, PP	Makespan	
2013	[15]	Definition 3, JPP	Makespan	Not specific to cyclic RFS
2014	[16]	Definition 2, PA	Total tardiness	Two-machine problem
2014	[29]	Definition 1, PP	Makespan	
2016	[28]	Definition 1, PP	Makespan	
2017	[23]	Definition 1, PP	Makespan	
2018	[27]	Definition 1, PP	Makespan	
2021	[24]	Definition 1, PP	Makespan, average completion times, total tardiness	Multi-objective performance measure
2023	[11]	Definition 1, PP	Makespan, maximum tardiness	Bi-objective performance measure
2023	[18]	Definition 2, PA	Value-at-risk of the makespan	Two-machine problem, stochastic processing times
2023	[26]	Definition 3, JPP	Makespan	Not specific to cyclic RFS, identical jobs

Table 1: Summary of the cyclic RPFS literature review.

There is also a fourth definition, for which the answers to Questions 1 and 2 are positive and negative, respectively. For this type, level passing is forbidden, but the job order may be different at different levels. We suggest to term this RPFS type as **Level Passing Prohibited** (LPP). Table 2 summarizes the four types of RPFS.

Permutation Type	Job Passing (Question 1)	Level Passing (Question 2)
Passing Prohibited (PP)	Not Allowed	Not Allowed
Level Passing Prohibited (LPP)	Allowed	Not Allowed
Job Passing Prohibited (JPP)	Not Allowed	Allowed
Passing Allowed (PA)	Allowed	Allowed

Table 2: Four possible permutation types in cyclic RPFS.

Example 1. An RFS problem with $n = 3$ jobs, $m = 3$ machines, $l = 2$ levels, and the following processing times:

$$P_1 = \begin{pmatrix} 4\,2\,4 & 2\,2\,8 \\ 4\,4\,2 & 4\,2\,8 \\ 8\,2\,2 & 6\,2\,2 \end{pmatrix}$$

Here, each line corresponds to a job, and each column to an operation according to its order in the flow. For example, the processing times of job j_1 in machines 1, 2, and 3 are $(4, 2, 4)$ at level l_1 and $(2, 2, 8)$ at level l_2.

Figure 1 shows potential schedules based on the four permutation types. Different colors represent different jobs (j_1 – red, j_2 – green, and j_3 – blue):

- PP type schedule with job order $(2, 1, 3)$ at the two levels.
- LPP type schedule with job order $(2, 1, 3)$ at the first level and $(1, 2, 3)$ at the second level.
- JPP type schedule with job order $(2, 1, 3)$ for both levels, but l_2 of j_2 (green) precedes l_1 of j_3 (blue).
- PA type schedule with job order $(2, 1, 3)$ at the first level and $(1, 2, 3)$ at the second level. In addition, l_2 of j_1 (red) precedes l_1 of j_3 (blue).

Figure 2 depicts a Venn diagram of the solution space of the different permutation types. As follows from the definitions and visually presented by the Venn diagram, the largest solution space is of the PA type with $(3 \cdot 2)!/(2!)^3 = 90$ possible permutations for Example 1. The smallest solution space is of the PP type with $3! = 6$ possible permutations. The intermediate types contain $(3!)^2 = 36$ and $5 \cdot 3! = 30$ possible permutations for the LPP and JPP types of Example 1, respectively.

3 Heuristic Issues

A classical three-machine permutation flow shop scheduling problem with the makespan objective is strongly NP-hard even without job reentrancy [14]. The problem Fm |

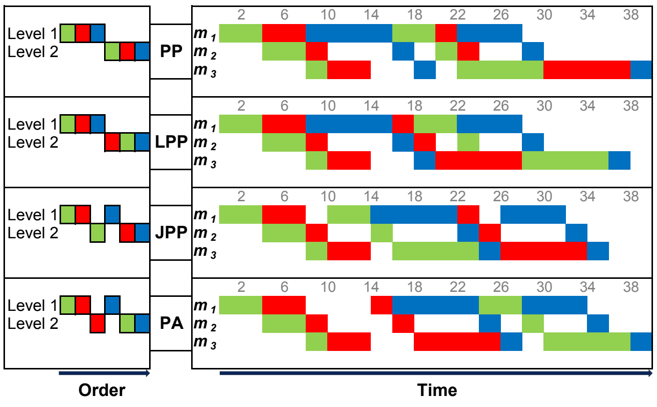

Fig. 1: Possible schedules of the four permutation types: passing prohibited (PP), level passing prohibited (LPP), job passing prohibited (JPP), passing allowed (PA).

reentrant | $C_{\max}$ is NP-hard, even for a two-machine problem [10]. Most studies on RFS problems thus focus on improving the computational efficiency of optimization algorithms (e.g., branch and bound) or developing efficient heuristics and metaheuristics. A common approach to devising a heuristic for RPFS is to adjust a (non-reentrant) flow shop heuristic to the RFS problem. Several studies applied this methodology while considering the PP [21] and PA [7] permutation types. The intermediate types – LPP and JPP – were not considered in this respect.

This section aims to show the relevance of the different permutation types to heuristics construction. We demonstrate this issue by examining Palmer's slope heuristic [20] as an initial step toward analyzing other commonly used PFS heuristics, such as CDS [2], NEH [19], and others. Palmer's slope heuristic has already been generalized for solving an RFS problem with makespan objective [21]. The obtained solution is of a PP permutation type. The generalization to the other permutation types is obtained by treating the processing times of each job as the processing times of a job in a non-reentrant flow shop (as if a machine at each level is a machine in itself). For each job, the slope index is calculated, and the jobs are arranged in descending order of the slope. When passing is prohibited, the job order is sufficient to determine the schedule. This simple procedure can be naturally generalized to the other permutation types by the calculation of slope indices for each of the sub-jobs. Treating sub-jobs is acceptable in generalizing flow shop heuristics to the reentrant with PA permutation type [7]. The generalization we consider here is to define an order of the sub-jobs according to their slopes, keeping in mind the constraints each type induces:

- For the LPP permutation type, we first arrange the sub-jobs of all the jobs of the first level according to its slope indices, from the largest to the smallest. We then arrange the sub-jobs of the second level according to its slope. We continue this, level after level, until the last level is reached and arranged.

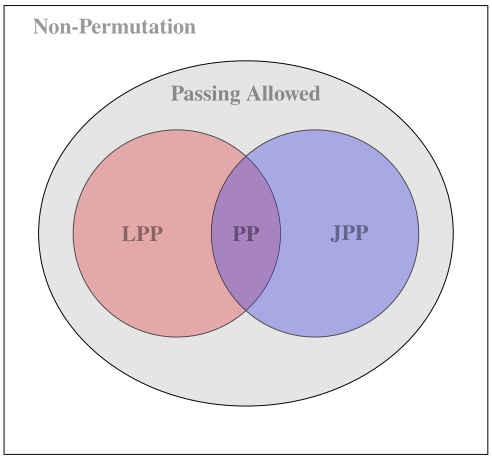

Fig. 2: A Venn diagram of the RPFS permutation types, including non-permutation RFS.

- For the PA permutation type, we arrange all the sub-jobs according to the slope indices while taking care of operation precedence, i.e., for each job, the sub-jobs of that job will be arranged in ascending order of levels.
- For the JPP permutation type, we regard a combination of the two types of slope indices. A job order is first determined according to the rule established for the PP type (considering the job slope indices). Then, the sub-jobs are arranged according to the sub-job indices while taking care of level precedence (like in the PA type), i.e., ensuring that the sub-job order is maintained within each job.

Once the rule for sub-job order (described above for each of the permutation types) is applied, a schedule is constructed by selecting each sub-job according to the order and inserting it after the previous sub-job. Another step that is generally considered in heuristics is that of solution improvement [13]. A standard improvement is obtained by sub-job swaps. The described heuristic, including some swap improvements, is shown in the following examples.

Consider Example 1 of Section 2. For this small example, following the above heuristic steps results in only two schedules for the four permutation types. The PP-type and LPP-type schedules are identical, with a makespan of 38. The JPP-type and the PA-type schedules are identical, with a makespan of 50. The example shows a major challenge in the generalization of heuristics based on the division into sub-jobs: how to avoid, or at least reduce, cases in which sub-jobs that belong to the same job are sequentially scheduled. In the second schedule, two such sequences greatly increase the makespan. By applying two simple swaps to avoid these sequences, a JPP-type schedule with a makespan of 36 can be obtained, see Figure 3.

Another challenge related to the construction of heuristics is related to tie-breaking, as demonstrated by Example 2.

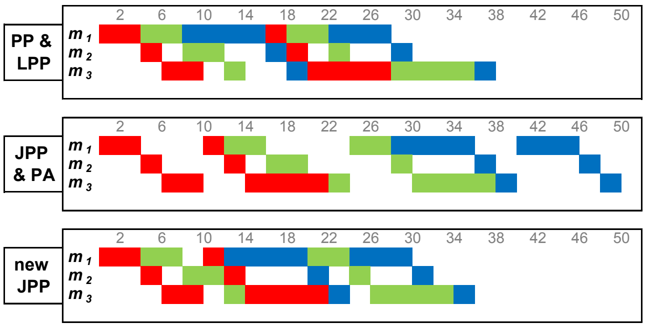

Fig. 3: Schedules obtained by the adjusted slope heuristic. The "new JPP" schedule is obtained from the JPP&PA schedule by two swaps: $j_1l_2 \leftrightarrow j_2l_1$ and $j_2l_2 \leftrightarrow j_3l_1$.

Example 2. An RFS problem with $n = 4$ jobs, $m = 3$ machines, $l = 3$ levels, and the following processing times:

$$P_2 = \begin{pmatrix} 6\,4\,2 & 8\,6\,4 & 6\,2\,2 \\ 8\,6\,8 & 6\,4\,2 & 2\,2\,6 \\ 4\,2\,4 & 6\,4\,6 & 4\,4\,8 \\ 8\,6\,8 & 4\,6\,4 & 8\,6\,4 \end{pmatrix}$$

Recall that each line corresponds to a job, and each column corresponds to an ordered operation.

In Example 2, several sub-jobs have an identical slope index. For the LPP type, 96 different sub-job orders can be obtained depending on the rule that breaks the tie. Finding a good tie-breaking rule is a known problem in flow shops [12] and can be significant for the quality of the solution. Figure 4 shows two possible LPP-type schedules. There is a tie between the schedules regarding the sub-job indices but the makespan is significantly different.

For both Examples 1 and 2, better solutions were obtained by allowing passing. In other scenarios, the strict PP type may be dominant. It is clear that a solution that allows passing is always at least as good as a solution that prohibits it, and a substantial challenge remains in finding good solutions by efficiently examining only a part of the solution space.

4 Discussion

It is known that there are scenarios for which the optimal solution is not a permutation schedule. This is true for $Fm||C_{max}$ compared to $Fm|prmu|C_{max}$ if $m > 3$ [10]. For the RFS problem, this is true even for two machines and the PA less-restricting permutation type [7]. Nevertheless, many real-world RFS manufacturing systems prefer to use permutation schedules because they offer greater ease of operation and control.

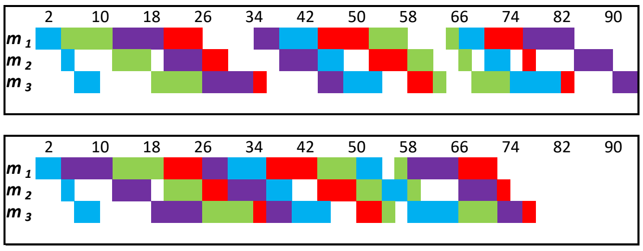

Fig. 4: Two LPP type schedules with ties regarding the sub-job slopes (j_1 – red, j_2 – green, j_3 – blue. and j_4 – purple)

In some cases, only permutation schedules are feasible because of the inflexibility of material handling systems or limited buffer space [16]. Theoreticians may prefer the permutation version as an algorithmically simplifying assumption. The permutation scenario was also adjusted to the reentrant case. However, the meaning of "permutation" has had several interpretations. We suggest to clarify the definition using the four permutation types.

Several heuristics have been proposed over the years to achieve a high-quality RPFS solution [21,7,8,17]. The types of permutations were only partially considered in previous work on RFS heuristics. We have shown a possible approach to extend Palmer's slope heuristic demonstrating the potential in regarding the four permutation types, as well as the challenges ahead. This only serves as a preview for further comprehensive research that will involve addressing larger problems and exploring other heuristics. A generalization of existing heuristic methods to each permutation type may provide a flexible and efficient framework for scheduling in RFS environments.

References

1. Baker, K.R.: Introduction to sequencing and scheduling (1974)
2. Campbell, H.G., Dudek, R.A., Smith, M.L.: A heuristic algorithm for the n job, m machine sequencing problem. Management science **16**(10), B–630 (1970)
3. Chen, J.S.: A branch and bound procedure for the reentrant permutation flow-shop scheduling problem. The International Journal of Advanced Manufacturing Technology **29**, 1186–1193 (2006)
4. Chen, J.S., Pan, J.C.H., Lin, C.M., et al.: Solving the reentrant permutation flow-shop scheduling problem with a hybrid genetic algorithm. International Journal of Industrial Engineering **16**(1), 23–31 (2009)
5. Chen, J.S., Pan, J.C.H., Wu, C.K.: Minimizing makespan in reentrant flow-shops using hybrid tabu search. The International Journal of Advanced Manufacturing Technology **34**, 353–361 (2007)
6. Chen, J.S., Pan, J.C.H., Wu, C.K.: Hybrid tabu search for re-entrant permutation flow-shop scheduling problem. Expert Systems with Applications **34**(3), 1924–1930 (2008)
7. Choi, S.W., Kim, Y.D.: Minimizing makespan on a two-machine re-entrant flowshop. Journal of the Operational Research Society **58**(7), 972–981 (2007)

8. Choi, S.W., Kim, Y.D.: Minimizing makespan on an m-machine re-entrant flowshop. Computers & Operations Research **35**(5), 1684–1696 (2008)
9. Danping, L., Lee, C.K.: A review of the research methodology for the re-entrant scheduling problem. International Journal of Production Research **49**(8), 2221–2242 (2011)
10. Emmons, H., Vairaktarakis, G.: Flow shop scheduling: theoretical results, algorithms, and applications, vol. 182. Springer Science & Business Media (2012)
11. Fasihi, M., Tavakkoli-Moghaddam, R., Jolai, F.: A bi-objective re-entrant permutation flow shop scheduling problem: minimizing the makespan and maximum tardiness. Operational Research **23**(2), 29 (2023)
12. Fernandez-Viagas, V., Framinan, J.M.: On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Computers & Operations Research **45**, 60–67 (2014)
13. Framinan, J.M., Gupta, J.N., Leisten, R.: A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society **55**, 1243–1255 (2004)
14. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of operations research **1**(2), 117–129 (1976)
15. Hinze, R., Sackmann, D., Buscher, U., Aust, G.: A contribution to the reentrant flow-shop scheduling problem. IFAC Proceedings Volumes **46**(9), 718–723 (2013)
16. Jeong, B., Kim, Y.D.: Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times. Computers & operations research **47**, 72–80 (2014)
17. Jing, C., Tang, G., Qian, X.: Heuristic algorithms for two machine re-entrant flow shop. Theoretical Computer Science **400**(1-3), 137–143 (2008)
18. Liu, L., Urgo, M.: Robust scheduling in a two-machine re-entrant flow shop to minimise the value-at-risk of the makespan: branch-and-bound and heuristic algorithms based on markovian activity networks and phase-type distributions. Annals of Operations Research pp. 1–24 (2023)
19. Nawaz, M., Enscore Jr, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega **11**(1), 91–95 (1983)
20. Palmer, D.S.: Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. Journal of the Operational Research Society **16**(1), 101–107 (1965)
21. Pan, J.H., Chen, J.S.: Minimizing makespan in re-entrant permutation flow-shops. Journal of the operational Research Society **54**, 642–653 (2003)
22. Pinedo, M.: Scheduling. theory, algorithms and systems. ISBN0-13-706757-7 (1995)
23. Qian, B., Li, Z.c., Hu, R.: A copula-based hybrid estimation of distribution algorithm for m-machine reentrant permutation flow-shop scheduling problem. Applied Soft Computing **61**, 921–934 (2017)
24. Rifai, A.P., Kusumastuti, P.A., Mara, S.T.W., Norcahy, R., Dawal, S.: Multi-operator hybrid genetic algorithm-simulated annealing for reentrant permutation flow-shop scheduling. ASEAN Engineering Journal **11**(3), 109–126 (2021)
25. Sauvey, C., Sauer, N.: Two neh heuristic improvements for flowshop scheduling problem with makespan criterion. Algorithms **13**(5), 112 (2020)
26. Shufan, E., Grinshpoun, T., Ikar, E., Ilani, H.: Reentrant flow shop with identical jobs and makespan criterion. International Journal of Production Research **61**(1), 183–197 (2023)
27. Wu, C.C., Liu, S.C., Cheng, T., Cheng, Y., Liu, S.Y., Lin, W.C.: Re-entrant flowshop scheduling with learning considerations to minimize the makespan. Iranian Journal of Science and Technology, Transactions A: Science **42**, 727–744 (2018)
28. Xu, J., Lin, W.C., Wu, J., Cheng, S.R., Wang, Z.L., Wu, C.C.: Heuristic based genetic algorithms for the re-entrant total completion time flowshop scheduling with learning con-

sideration. International Journal of Computational Intelligence Systems **9**(6), 1082–1100 (2016)

29. Xu, J., Yin, Y., Cheng, T.C.E., Wu, C.C., Gu, S.: A memetic algorithm for the re-entrant permutation flowshop scheduling problem to minimize the makespan. Applied Soft Computing **24**, 277–283 (2014)
30. Yu, T.S., Pinedo, M.: Flow shops with reentry: Reversibility properties and makespan optimal schedules. European Journal of Operational Research **282**(2), 478–490 (2020)