Reentrant Permutation Flow Shop Problems with the Total Completion Time Objective

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. Permutation schedules in reentrant flow shops can be defined in several ways depending on job-passing and level-passing constraints across the sub-jobs. This gives rise to four permutation types. In this paper, we investigate the reentrant permutation flow shop with the objective of total completion time for the four permutation types. We show that allowing level passing is more effective at reducing the objective function value than allowing job passing. We conduct a systematic analysis of small instances to demonstrate this observation. We also address larger instances using the simulated annealing metaheuristic. The computational results demonstrate the strength of level passing with respect to the total completion time objective. In particular, the permutation type that allows level passing while prohibiting job passing offers a good trade-off for obtaining high-quality solutions in a reduced solution space.

Keywords: Permutation Flow Shop · Reentrant Flow Shop · Permutation Types · Total Completion Time · Simulated Annealing

1 Introduction

A flow shop is a scheduling environment in which all jobs visit a set of machines in the same order. A well-studied special case is the *permutation* flow shop (PFS), in which the job sequence is identical across all machines. In a standard flow shop, each job visits each machine at most once. However, in many real-world manufacturing systems, jobs must return to the same machine multiple times during processing. Such problems are referred to as *reentrant* flow shops (RFS). A particular class of RFS is the *cyclic* RFS, in which each job passes through all machines multiple times [3]. A cyclic RFS is characterized by n jobs, m machines, and l levels. Each level represents a full pass of a job through the machines in the order $(1, 2, \ldots, m)$. The processing of a job at a specific level is called a sub-job. Graves et al. [7] first introduced the RFS problem, and it has since been studied extensively, as reviewed by von Aspern et al. [1].

The notion of permutation has been extended to the RFS problem, leading to the *reentrant permutation flow shop* (RPFS), primarily in the cyclic setting.

In previous work [12, 13], we defined four RPFS permutation types based on two scheduling constraints:

- **Level passing:** whether a job may proceed to a higher level before all sub-jobs at the current level are completed. If so, level passing is allowed; otherwise, it is prohibited.
- **Job passing:** whether the job order within a given level must be identical across all levels. If so, job passing is prohibited; otherwise, it is allowed.

The four permutation types are defined according to whether level passing and job passing are allowed or prohibited: Passing Prohibited (PP), Level Passing Prohibited (LPP), Job Passing Prohibited (JPP), and Passing Allowed (PA). In particular, LPP prohibits level passing while allowing job passing, whereas JPP prohibits job passing while allowing level passing. In previous work, we showed that, for the makespan objective, the freedom afforded by allowing job passing is more effective than that provided by level passing [13]. In particular, LPP proved to be a successful permutation type in balancing the objective (high solution quality) and effort (low solution space).

Most previous studies on reentrant flow shop problems have focused on the makespan ($C_{\max}$) objective. Conversely, the *total completion time* objective has received considerably less attention, particularly in the RPFS setting. Here, we investigate the total completion time objective in cyclic RPFS under four permutation types. The problem is denoted by $F_m|\text{cyclic-reentrant}, \text{perm-}X|\sum C_j$, where $X \in \{\text{PP}, \text{LPP}, \text{JPP}, \text{PA}\}$, using the three-field notation [6]. We focus on small instances, for which optimal solutions can be obtained, analyze their properties, and examine how they differ from those under the makespan objective. In addition, we propose a simulated annealing (SA) metaheuristic to address larger instances.

Small illustrative instances show that neither level passing nor job passing uniformly dominates the other; depending on the processing times, each type of freedom may be either crucial or negligible. This was also the case for the makespan objective. Nevertheless, our computational study on both small and large instances reveals a clear overall trend: For the $\sum C_j$ objective, the flexibility associated with level passing tends to have a stronger impact than that associated with job passing. This contrasts with our findings for the makespan objective, where job passing was more influential. In particular, JPP (rather than LPP) provides a favorable trade-off between solution quality and computational efficiency. This can be intuitively explained by the nature of the $\sum C_j$ goal, which aggregates the completion times of all jobs and therefore favors schedules that enable earlier finishing of individual jobs. Such early finishing has little impact on the makespan, which is determined solely by the completion time of the final job.

The remainder of the paper is organized as follows. Subsection 1.1 reviews relevant literature on the total completion time objective in flow shop and RFS settings. Section 2 analyzes small instances, while Section 3 presents an SA–based approach for larger instances. A discussion in Section 4 concludes the paper.

1.1 Related Work

The PFS problem with the total completion time objective is NP-hard already for two machines [5]. Therefore, a variety of heuristics have been proposed, including the RZ heuristic [11], the NEH-based flowtime heuristic [4], and the LR heuristic [10]. The literature on cyclic RPFS under the total completion time objective remains limited. To the best of our knowledge, only a few closely related studies have considered adapting classical PFS heuristics to reentrant settings. One such study is of Xu et al. [15], who consider an RPFS with a position-based learning effect using a single job sequence across all levels, which can be interpreted as a structure corresponding to the PP type in our framework. They apply classical heuristics, including RZ and NEH, further enhanced by local search and genetic algorithm procedures. Jing et al. [9] studied a model equivalent to the PA structure and observed that, since the objective is to minimize the total completion time, it is beneficial to process a job's sub-jobs as close together as possible so that the job completes and leaves the system early. However, scheduling sub-jobs too closely may result in significant idle time due to precedence constraints between levels. Accordingly, they propose a heuristic algorithm based on a k-insertion technique, which iteratively improves sub-job sequences by controlling the distance between levels. For the two-machine case, they also derive a lower bound based on Johnson's rule. Aspern et al. [1] study RFS under the total weighted completion time objective, assuming unit processing times. They propose the Least Remaining Loops (LRL) rule, which is optimal for the unweighted case, and analyze its weighted extension. They also develop a pseudo-polynomial-time algorithm and a fully polynomial-time approximation scheme for a fixed number of machines. These works highlight different aspects of the total completion time objective in reentrant settings, yet a comprehensive treatment of cyclic RPFS across all permutation types remains largely unexplored.

2 Analysis of Small Instances

Following the approach for the makespan objective [13], we analyze the impact of the different permutation types on solution quality under the total completion time objective. Let PP, LPP, JPP, and PA, denote the solution space for the corresponding permutation types. Let $\sum C_j(OPT, X)$ denote the optimal total completion time for $X \in \{\text{PP}, \text{LPP}, \text{JPP}, \text{PA}\}$. By definition[1], $\text{PP} \subset \text{LPP}, \text{JPP} \subset \text{PA}$. Therefore, $\sum C_j(OPT, \text{PA}) \leq \sum C_j(OPT, \text{LPP}) \leq \sum C_j(OPT, \text{PP})$ and similarly $\sum C_j(OPT, \text{PA}) \leq \sum C_j(OPT, \text{JPP}) \leq \sum C_j(OPT, \text{PP})$. In some instances, equalities may replace some inequalities. When $\sum C_j(OPT, \text{PP}) = \sum C_j(OPT, \text{PA})$ for a given instance, we say that PP dominates PA for that instance, because PP achieved the same objective as PA even though PP is a strict subset of PA. Generalizing this, if $X \subset Y$ and $\sum C_j(OPT, X) = \sum C_j(OPT, Y)$, we say that X dominates Y. If, on the other hand, $\sum C_j(OPT, Y) < \sum C_j(OPT, X)$,

[1] The notation $\subset$ is used here to indicate a proper subset.

we say that Y dominates X. This dominance relation emphasizes the tension between quality and efficiency. An additional measure for the quality of a permutation type for a given instance is the optimality gap, defined as

$$\text{Optimality gap}(X) \;=\; \frac{\sum C_j(OPT, X) - \sum C_j(OPT, \text{PA})}{\sum C_j(OPT, \text{PA})} \times 100\,.$$

2.1 Simple Examples

Examples 1–4 below demonstrate that each of the four permutation types may dominate the other permutation types. In each example, every set among PP, LPP $\setminus$ PP (strict LPP), JPP $\setminus$ PP (strict JPP), and PA $\setminus$ (LPP $\cup$ JPP) (strict PA) admits a unique optimal solution.

Example 1. Consider a problem with 2 jobs, 2 machines, and 2 levels with the following processing times (each matrix line corresponds to a job, each column to a machine in a certain level, where different levels are separated by a large space):

$$\begin{pmatrix} 1 & M & 1\,1 \\ M & 1 & 1\,1 \end{pmatrix}$$

Assume that M is a sufficiently large number. The optimal schedule is a PP schedule, with objective value $\Sigma C_j = 2M + 7$. It is optimal already for $M = 2$. Figure 1 (left) shows the Gantt chart of the optimal schedule with $M = 10$. Different jobs are shown by different colors. Stars indicate the completion time of each job. Figure 1 (right) shows the sub-job order, emphasizing the nature of the permutation type.

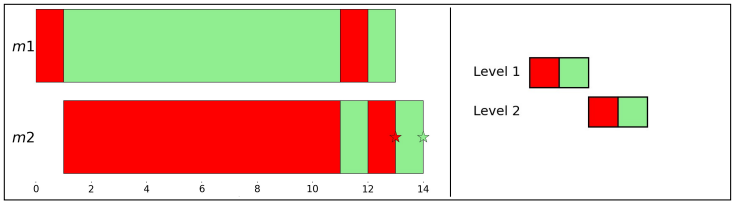


Fig. 1: Gantt chart (left) and sub-job order (right); PP-dominated.

In Example 1, PP dominates the other permutation types. Optimality can be proved by brute force, but it follows from the simultaneous processing of operations with long processing times. This is also true for the following examples of optimal schedules, which illustrate the respective dominance of LPP, JPP, and PA. These solutions are also optimal for the makespan objective [13].

Example 2. Consider a problem with 2 jobs, 2 machines, and 2 levels with the following processing times:

$$\begin{pmatrix} 1 & M & M & 1 \\ M & 1 & 1 & M \end{pmatrix}$$

The optimal schedule, shown in Figure 2, is a strict LPP schedule, with objective value $\sum C_j = 4M + 7$.

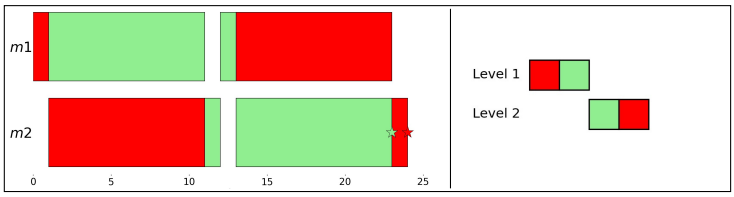


Fig. 2: Gantt chart (left) and sub-job order (right); LPP-dominated.

Example 3. Consider a problem with 3 jobs, 2 machines, and 2 levels with the following processing times:

$$\begin{pmatrix} 1 & M & 1 & M \\ M & 1 & 1 & M \\ M & 1 & M & 1 \end{pmatrix}$$

The optimal schedule, shown in Figure 3, is a strict JPP schedule, with objective value $\sum C_j = 8M + 9$.

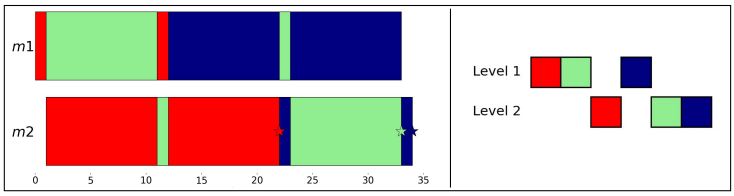


Fig. 3: Gantt chart (left) and sub-job order (right); JPP-dominated.

Example 4. Consider a problem with 3 jobs, 2 machines, and 2 levels with the following processing times:

$$\begin{pmatrix} 1 & 2M & 1 & M \\ 2M & 1 & 2M & 1 \\ M & 1 & 1 & 2M \end{pmatrix}$$

The optimal schedule, shown in Figure 4, is a strict PA schedule, with objective value $\sum C_j = 13M + 11$.

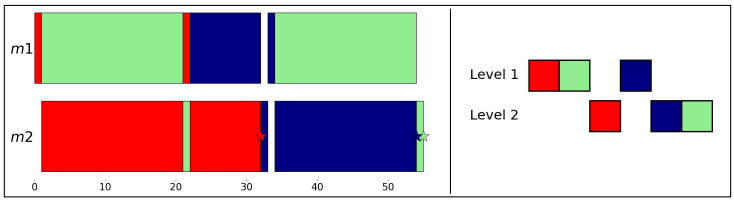


Fig. 4: Gantt chart (left) and sub-job order (right); PA-dominated.

2.2 Small Instances: Varying n and l

We consider 18 experimental sets, each containing 50 instances. Optimal solutions were obtained by exhaustive search for all instances, by enumerating all feasible permutations and evaluating their total completion times. The number of machines is fixed at $m = 5$, with n and l varying. The experimental sets include $n = 2$ with $l = 2, \ldots, 11$, $n = 3$ with $l = 2, \ldots, 5$, $n = 4$ with $l = 2, 3$, and $n = 5, 6$ with $l = 2$. We denote an experimental set by $n \times m \times l$. These instances were originally generated for experiments on the makespan objective [13]; details on their generation and full results can be found in that work. Nonetheless, we summarize key findings from the makespan experiments [13] for comparison with our total completion time results.

The Pareto-dominant experimental sets, those corresponding to the largest combinations of n and l, are $2 \times 5 \times 11$, $3 \times 5 \times 5$, $4 \times 5 \times 3$, and $6 \times 5 \times 2$. The results for these sets are presented in Figure 5 as Venn diagrams, one for each experimental set. For each set, we report the number of instances in which a solution space is dominant over the others. For example, among the $2 \times 5 \times 11$ instances, PP dominates in 15 cases, LPP in 9, JPP in 24, and PA in 2. The total may exceed 50 (in other experimental sets) because, in some instances, both LPP and JPP dominate. In addition to these counts shown in bold, we report in parentheses the size of the solution space (i.e., the number of feasible schedules) for each permutation type.

The results for all 18 experimental sets are presented in Figures 6 and 7, showing the effect of varying n and l, respectively. As n or l increases, the number of instances in which PP dominates decreases and shifts toward the other sets, initially to LPP, then increasingly to JPP, and eventually to PA. These findings differ from the makespan results, in which PP remained dominant more frequently and declined more gradually as n and l increased, with a stronger shift toward LPP and a weaker shift toward JPP.

Table 1 reports the average optimality gap across the Pareto-dominant experimental sets and the maximum optimality gap observed among the 50 instances of each set. The largest average optimality gap is observed in PP, followed by LPP and JPP. Compared to the makespan results, the average optimality gap for PP and LPP is higher, and for JPP it is lower. The largest average optimality gap observed in the set $6 \times 5 \times 2$ is 1.25% for the makespan objective, while it reaches 5.94% for the total completion time objective. The JPP type exhibits lower average optimality gaps than the LPP type across all examined sets, even

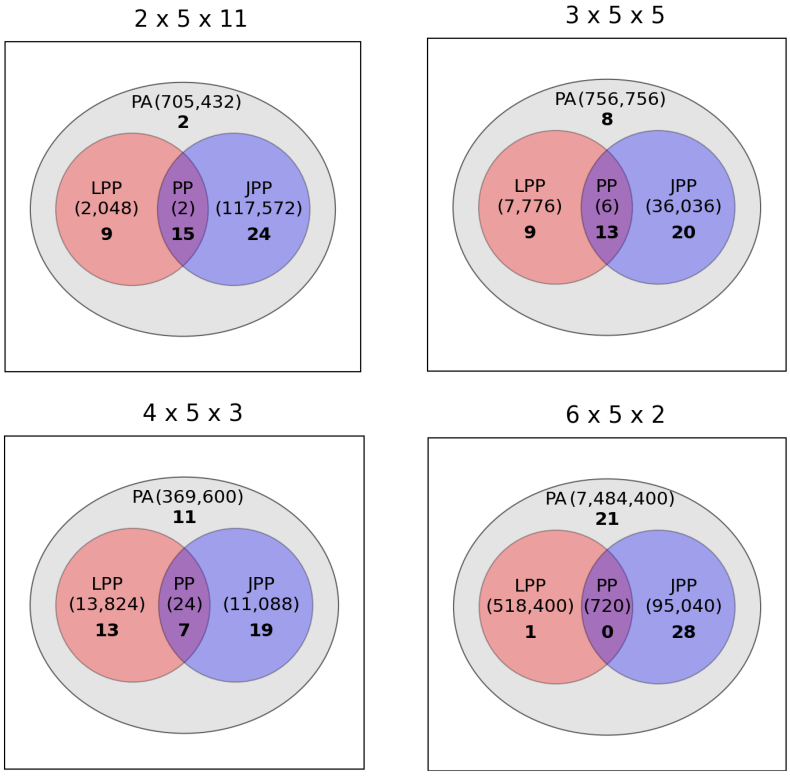
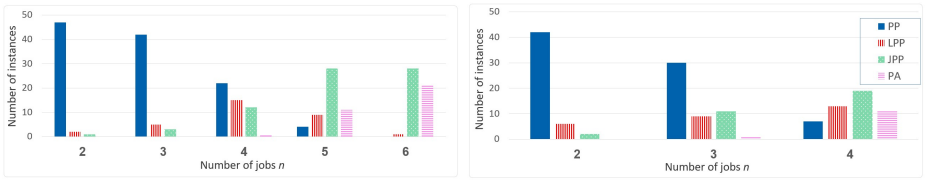


Fig. 5: Venn diagrams of the solution spaces showing the number of instances each space dominates (bold), and their sizes (parentheses).

in cases where its solution space is smaller. In general, the average optimality gap within each permutation type tends to increase as n or l increases.

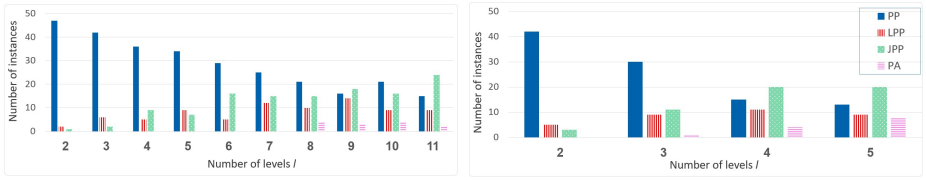
Another measure is the number of optimal solutions per instance, reported in Table 2 for the Pareto-dominant sets. For each permutation type, we report the average number of optimal solutions per instance and the percentage of instances with a single optimal solution. The average number of optimal solutions per instance is very small relative to the size of the solution space, and substantially lower than those observed for the makespan objective. For example, consider the PA type in the $6 \times 5 \times 2$ set. Its solution space contains approximately 7.5 million feasible schedules, while the average number of optimal solutions per instance is only 1.12, with 88% of the instances having a unique optimal solution.

These observations highlight the increasing difficulty of identifying optimal solutions as the problem size grows. For larger RPFS instances, exact solution methods become computationally impractical, and other approaches such as heuristics or metaheuristics are required. Based on our previous findings for the makespan objective, extensions of classical PFS heuristics to the RPFS setting



(a) $l = 2$ (b) $l = 3$

Fig. 6: Number of instances in which PP, LPP, JPP, or PA dominates, for $m = 5$, as a function of n.



(a) $n = 2$ (b) $n = 3$

Fig. 7: Number of instances in which PP, LPP, JPP, or PA dominates, for $m = 5$, as a function of l.

were shown to be less effective as the problem size increases. Therefore, rather than directly adapting such heuristics, we adopt a metaheuristic approach and apply simulated annealing (SA) to address the total completion time objective.

3 Metaheuristic Results

In this section, we show that trends observed in small instances also hold in larger ones. To systematically explore the solution space, we employ SA, which is an effective metaheuristic for exploring large search spaces and producing high-quality solutions to complex optimization problems. Specifically, SA has been widely used in flow shop scheduling, e.g., [2]. We previously considered an SA approach for RPFS to address the makespan objective [14]; here, we apply it to the $\sum C_j$ objective.

3.1 Simulated Annealing for RPFS

Three standard neighborhood operators from the PFS setting [8] were extended to the four RPFS types as described below. These include:

1. Adjacent swap: interchange two adjacent jobs.
2. Global swap: interchange two jobs at arbitrary positions.
3. Insert: remove a job and reinsert it elsewhere.

Their extensions to RPFS were:

	$2 \times 5 \times 11$		$3 \times 5 \times 5$		$4 \times 5 \times 3$		$6 \times 5 \times 2$	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max
PP	1.39	6.31	1.59	9.51	2.27	7.03	5.94	12.64
LPP	0.71	3.68	0.89	5.40	1.20	6.35	4.34	10.66
JPP	0.29	3.48	0.57	3.66	0.67	4.73	0.35	1.98

Table 1: Average and maximum optimality gaps (%).

	$2 \times 5 \times 11$		$3 \times 5 \times 5$		$4 \times 5 \times 3$		$6 \times 5 \times 2$	
	Avg	Unique (%)	Avg	Unique (%)	Avg	Unique (%)	Avg	Unique (%)
PP	1.00	100	1.00	100	1.04	96	1.10	90
LPP	1.06	94	1.38	72	1.54	70	2.30	50
JPP	1.06	94	1.16	90	1.16	86	1.12	88
PA	1.04	96	1.28	80	1.32	82	1.12	88

Table 2: Average number of optimal solutions per instance (Avg) and percentage of instances with a unique optimal solution (Unique).

- **PP:** Applied exactly as in PFS, since neither job passing nor level passing is allowed.
- **LPP:** Applied independently within each level, either sequentially across all levels (level-by-level) or on a randomly chosen level.
- **JPP:** Applied to the sub-job sequence while preserving precedence (level $k+1$ cannot precede level k) and maintaining a consistent job order across levels. To explore job orders, an additional adjacent swap was applied to the job sequence, inducing the corresponding reordering of all sub-jobs, yielding four neighborhood structures.
- **PA:** Applied to the sub-job sequence subject only to precedence constraints, providing the most flexible neighborhoods.

3.2 Algorithm Parameters and Move Selection

We used an initial temperature $T_0 = 100$, a final temperature $T_{\text{final}} = 0.1$, and a cooling rate $\alpha = 0.98$. At each temperature, $10 \cdot n$ neighbors were evaluated for the PP type and the level-by-level LPP, and $5 \cdot n \cdot l$ for the random-level LPP, JPP, and PA. The initial sequence was generated as a random permutation.

For PP, LPP, and PA, moves were chosen uniformly at random. For JPP, moves followed a (0.3, 0.3, 0.3, 0.1) distribution, with the job-order move assigned a lower probability due to its stronger structural effect. We applied a temperature-dependent scheme: at high temperatures, selection followed the base distribution; as temperature decreased, the probabilities of global swap and insert moves decreased linearly, while local moves increased, shifting from exploration to exploitation. The job-order move probability remained fixed (0.1).

3.3 Experimental Results

We first evaluate the performance of SA relative to the optimal solutions of the small instances (Subsection 2.2). Let $\sum C_j(\mathrm{SA}, X)$ denote the total completion time computed by SA for permutation type $X \in \{\mathrm{PP}, \mathrm{LPP}, \mathrm{JPP}, \mathrm{PA}\}$. The performance gap relative to optimal solutions is

$$\text{Performance gap}(X) = \frac{\sum C_j(\mathrm{SA}, X) - \sum C_j(\mathrm{OPT}, X)}{\sum C_j(\mathrm{OPT}, X)} \times 100.$$

The results are presented in Table 3. For each Pareto-dominant set, we report the percentage of instances for which SA attained the optimal objective function for each permutation type, along with the average and maximum performance gaps. The results indicate that the proposed SA approach performed effectively across all permutation types, consistently achieving high optimality rates and very small performance gaps. In particular, for the PP and LPP types, the method attained optimal solutions in nearly all instances. For the JPP and PA types, a slight performance degradation was observed for the $6 \times 5 \times 2$ set. Nevertheless, even in this case, the obtained solutions remained close to the optimal solutions, as reflected by the relatively small average and maximum performance gaps. These results suggest that the proposed SA mechanism effectively balances exploration and exploitation under the total completion time objective.

	$2 \times 5 \times 11$			$3 \times 5 \times 5$			$4 \times 5 \times 3$			$6 \times 5 \times 2$		
	Opt	Avg	Max	Opt	Avg	Max	Opt	Avg	Max	Opt	Avg	Max
PP	100	0.00	0.00	100	0.00	0.00	100	0.00	0.00	100	0.00	0.00
LPP	100	0.00	0.00	100	0.00	0.00	100	0.00	0.00	98	<0.01	0.10
JPP	100	0.00	0.00	100	0.00	0.00	98	0.01	0.53	90	0.15	2.25
PA	100	0.00	0.00	98	0.01	0.36	98	0.01	0.26	66	0.17	1.48

Table 3: SA performance: percentage of optimal solutions, average gap, and maximum gap (%).

Since the optimal solutions are unknown for large instances, performance is measured relative to the *best* solution found by SA for each instance. In this setting, methods with smaller search spaces may occasionally yield better solutions. Let $\sum C_j(\mathrm{SA}, \mathrm{BEST})$ denote the best (minimal) total completion time computed by SA for an instance. The performance gap is defined as

$$\text{Performance gap}(X) = \frac{\sum C_j(\mathrm{SA}, X) - \sum C_j(\mathrm{SA}, \mathrm{BEST})}{\sum C_j(\mathrm{SA}, \mathrm{BEST})} \times 100\,.$$

We generated 50 instances for a base configuration of $10 \times 5 \times 3$ and evaluated the performance gap. We then considered two types of variations: (i) varying the number of jobs $n \in \{5, 10, 15\}$ while keeping $m = 5$ and $l = 3$, and (ii) varying the number of levels $l \in \{2, 3, 4\}$ while keeping $n = 10$ and $m = 5$.

The obtained performance gaps are presented in Table 4. In the $5 \times 5 \times 3$ experiments, LPP attains the best solution in only 5 instances, including 2 ties with either PA or JPP. In all other instances across all experimental sets, the best solutions are obtained by JPP or PA. Moreover, the average gap of PP and LPP is consistently larger than that of JPP and PA, and increases with instance size, reaching values significantly higher than those observed for the makespan objective. Finally, JPP achieves performance comparable to PA, likely because its solution space strikes a balance: it is large enough to contain high-quality solutions, yet small enough to allow effective exploration by the SA search.

	$m = 5,\ l = 3$		
	$5 \times 5 \times 3$	$10 \times 5 \times 3$	$15 \times 5 \times 3$
PP	3.45	16.53	25.67
LPP	2.34	13.31	22.33
JPP	0.37	0.49	0.69
PA	0.42	1.15	0.57

(a) Varying n

	$n = 10,\ m = 5$		
	$10 \times 5 \times 2$	$10 \times 5 \times 3$	$10 \times 5 \times 4$
PP	15.05	16.53	16.80
LPP	11.97	13.31	13.70
JPP	1.03	0.49	0.45
PA	0.43	1.15	1.16

(b) Varying l

Table 4: Average performance gap (%) based on the base configuration $10 \times 5 \times 3$: (a) varying the number of jobs n, and (b) varying the number of levels l.

4 Discussion

The classification of permutation types highlights the role of two sources of flexibility: changes in job order across levels and passing between levels. While increased flexibility enlarges the search space and may improve solution quality, this is not guaranteed. In particular, both exact and heuristic approaches may benefit from restricting the search to smaller spaces, where high-quality solutions can sometimes be identified more effectively. The relative importance of these two types of flexibility differs between objectives. For makespan, improvements are mainly driven by allowing changes in job order across levels [13]. In contrast, for the total completion time objective considered in this work, the dominant effect arises from allowing jobs to overtake one another, enabling earlier completion without waiting for other jobs.

The computational results for small instances confirm that each solution space may be dominant. Nevertheless, dominance is observed more frequently for JPP, which often provides high-quality solutions even when it is not dominant over PA. This trend persists in the larger instances. In this setting, JPP often yields competitive and, in many cases, superior solutions.

These findings suggest that JPP provides an effective balance between flexibility and search complexity. Further computational study is needed to better understand the impact of problem parameters and to determine whether the observed trends reflect structural properties of the solution spaces or limitations of

the search. In particular, it is natural to ask whether the tendency to prioritize shorter jobs, which is well known in total completion time problems, also plays a significant role in the RPFS setting.

References

1. Aspern, M.v., Buld, F., Klein, N., Pinedo, M.: Flow shops with reentry: The total weighted completion time objective. Journal of Scheduling pp. 1–16 (2025)
2. Chen, R.M., Sandnes, F.E.: Flow shop scheduling based on a novel cooling temperature driven simulated annealing algorithm. Scientia Iranica. Transaction B, Mechanical Engineering **22**(4), 1545 (2015)
3. Emmons, H., Vairaktarakis, G.: Flow shop scheduling: theoretical results, algorithms, and applications, vol. 182. Springer Science & Business Media (2012)
4. Framinan, J.M., Leisten, R., Ruiz-Usano, R.: Comparison of heuristics for flowtime minimisation in permutation flowshops. Computers & Operations Research **32**(5), 1237–1254 (2005)
5. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of operations research **1**(2), 117–129 (1976)
6. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of discrete mathematics, vol. 5, pp. 287–326. Elsevier (1979)
7. Graves, S.C., Meal, H.C., Stefek, D., Zeghmi, A.H.: Scheduling of re-entrant flow shops. Journal of operations management **3**(4), 197–207 (1983)
8. Hurkała, J., Hurkała, A.: Effective design of the simulated annealing algorithm for the flowshop problem with minimum makespan criterion. Journal of Telecommunications and Information Technology pp. 92–98 (2012)
9. Jing, C., Huang, W., Tang, G.: Minimizing total completion time for re-entrant flow shop scheduling problems. Theoretical Computer Science **412**(48), 6712–6719 (2011)
10. Liu, J., Reeves, C.R.: Constructive and composite heuristic solutions to the $p||\sum c_i$ scheduling problem. European Journal of Operational Research **132**(2), 439–452 (2001)
11. Rajendran, C., Ziegler, H.: An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Journal of Operational Research **103**(1), 129–138 (1997)
12. Segal, I., Grinshpoun, T., Ilani, H., Shufan, E.: The meaning of permutation in reentrant permutation flow shop problems. In: PATAT 2024: 14th International Conference on the Practice and Theory of Automated Timetabling. pp. 99–109 (2024)
13. Segal, I., Grinshpoun, T., Ilani, H., Shufan, E.: Reentrant permutation flow shop problems: definitions and heuristics. Journal of Scheduling (2026). https://doi.org/10.1007/s10951-026-00873-4
14. Segal, I., Grinshpoun, T., Ilani, H., Shufan, E.: Using Simulated Annealing for Solving Reentrant Permutation Flow Shop Problems. In: Proceedings of the 20th International Workshop on Project Management and Scheduling (PMS 2026). pp. 66–69 (2026)
15. 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 consideration. International Journal of Computational Intelligence Systems **9**(6), 1082–1100 (2016)