Scheduling in parallel machines with two servers: the restrictive case

Rachid Benmansour[1][0000−0003−2553−4116] and Angelo Sifaleras[2][0000−0002−5696−7021]

[1] National Institute of Statistics and Applied Economics (INSEA), SI2M Laboratory, Rabat, Morocco
r.benmansour@insea.ac.ma
[2] Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Str., Thessaloniki 54636, Greece
sifalera@uom.gr

Abstract. In this paper we study the Parallel machine scheduling problem with Two Servers in the Restrictive case (PTSR). Before its processing, the job must be loaded on a common loading server. After a machine completes processing one job, an unloading server is needed to remove the job from the machine. During the loading, respectively the unloading, operation, both the machine and the loading, respectively the unloading, server are occupied. The objective function involves the minimization of the makespan. A Mixed Integer Linear Programming (MILP) model is proposed for the solution of this difficult problem. Due to the NP-hardness of the problem, a Variable Neighborhood Search (VNS) algorithm is proposed. The proposed VNS algorithm is compared against a state-of-the-art solver using a randomly generated data set. The results indicate that, the obtained solutions computed in a short amount of CPU time are of high quality. Specifically, the VNS solution approach outperformed IBM CPLEX Optimizer for instances with 15 and 20 jobs.

Keywords: Scheduling · Parallel machine · Mixed integer programming · Variable neighborhood search · Single server.

1 Introduction

Sequencing and scheduling decisions are crucial in manufacturing and service industries. Scheduling jobs on parallel machines consists of determining the starting time of each job and the machine that will process this job. The problem has a myriad of applications in logistics, manufacturing, and network computing etc. The literature on this subject is abundant as it is for the problem of parallel machine scheduling problem with single server ([1, 2, 6, 12]).

Formally the problem of minimizing the makespan C_{max} on the parallel machine with a single server is denoted by $Pm, S1|p_i, s_i|C_{max}$. In this notation, m represents the number of machines, $S1$ represents the server, and p_i, s_i represent the processing time and setup time (or loading time) of job i, respectively.

Please cite this paper as:

Benmansour R. and Sifaleras A., "Scheduling in parallel machines with two servers: The restrictive case", *Variable Neighborhood Search* (ICVNS 2021), Lecture Notes in Computer Science, Springer , Cham, Vol. 12559, pp. 71-82, 2021.

The final publication is available at Springer via https://doi.org/10.1007/978-3-030-69625-2_6

Considering this problem, Liu et al. [13] studied the objective of minimizing the total weighted job completion time. They proposed a branch-and-bound algorithm, a lower bound, and dominance properties.

In [12] the authors proposed two mixed integer programming formulations for the problem with several machines. They proposed also a hybrid heuristic algorithm combining Simulated Annealing (SA) and Tabu Search (TS) to minimize the total server waiting time. Recently El Idrissi et al. [7] proposed two additional mixed integer programming formulations for the same problem with better performance given especially by the time-indexed variables formulation.

In the paper of Torjai and Kruzslicz [17], the authors consider the situation where the shared server is used to unload the jobs. As an application, the authors studied a biomass truck scheduling problem. The trucks represent the machines in charge of delivering biomass from different locations to a single refinery operating a single server.

The problem of considering both loading and unloading operations in order to minimize the makespan was studied by Jiang et al. [11]. In their work the authors considered only two parallel machine and a single server that is capable of doing the loading and unloading operations. In addition to these assumptions, the authors considered the non-preemptive case in which the loading and unloading durations are both equal to one time unit. Given the NP-hardness nature of the problem, they applied a List Scheduling (LS) and Longest Processing Time (LPT) to solve this problem. Also they showed that LPT and LS LPT have tight worst-case ratios of 4/3 and 12/7, respectively.

Some authors considered the problem with several servers. Ou et al. [15] studied the parallel machine scheduling with multiple unloading servers in order to minimize the total completion time of the jobs. As an application the authors cite milk run operations of a logistics company that faces limited unloading docks at the warehouse. The authors show that, the shortest-processing-time-first (SPT) algorithm has a worst-case bound of 2. They also provide heuristics and a branch-and-bound algorithm to solve the problem.

In [18] the authors studied the problem of scheduling non preemptively a set of jobs identical parallel machines. Each job has to be loaded, on a given machine, by one of multiple servers available. The authors show that, the problem $Pm, Sk|s_i = 1|C_{max}$ is binary NP-hard and that the problem $Pm, S(m-1)|s_i = 1|C_{max}$ can be solved by a pseudo-polynomial algorithm. For a fixed number of machines and servers the problem is unary NP-hard when considering maximum lateness minimization.

As seen above, there are several articles dealing with the problem of parallel machine scheduling with loading and unloading operations. In some cases only one server was considered, and in other cases several servers were considered. Also several researchers considered only the loading or unloading operations whereas other researchers considered that the server can do both the loading operation and the unloading operation.

To the best of our knowledge, the case where two servers are available has not been studied before. One server is dedicated only for loading jobs on the

machines and the other server is dedicated for unloading them from the machines. This problem is NP-hard as it is more difficult that the special case $P2, S1|p_j, s_j|C_{max}$ which is NP-hard (cf. [4]). The research contributions of this work are summarized as follows:

- This paper considers the parallel machine scheduling problem with loading and unloading servers.
- The general problem with restrictive and non-preemptive case is considered for the first time.
- An efficient VNS algorithm is proposed to solve this problem.

The remaining of the paper is organized as follows. The description of the problem and an illustrative example are given in Section 2. Section 3 presents the mathematical formulation of the restrictive model. Section 4 provides a VNS method for the solution of the PTSR. Finally, Sections 5 and 6 present our experimental results and draw some conclusions, respectively.

2 Problem Description

This paper considers the parallel machine scheduling problem with loading and unloading servers. In this problem, we consider two machines and a set $\Omega_1 = \{1, 2, \ldots, n\}$ of n independent jobs with integer processing times have to be processed non-preemptively on a set of parallel machines with two servers. The first server is dedicated to the loading of jobs on the machines and the second server realizes the unloading of jobs immediately after their execution. During the loading (respectively unloading) operation, both the machine and the loading server (respectively unloading server) are occupied and after this operation, the server becomes available for loading (respectively unloading) the next job. It is assumed that, the jobs are simultaneously available for processing at the beginning of the scheduling horizon, and that their processing times are fixed and known in advance. The objective function in the *PTSR* problem consists of minimizing the makespan.

2.1 Numerical example

Let's assume we are given an instance with two parallel machines M1 and M2, eight jobs, and two servers. One server is used to load the jobs on one of the machines (denoted as **L**) and the other is used to unload them (denoted as **U**). The other data are displayed in the following Table 2.1. For each job i, p_i, l_i, and u_i represent the processing time, the loading time, and the unloading time of this job, respectively.

The optimal objective function value can be obtained by solving the MIP formulation described in Section 3. IBM CPLEX Optimizer v12.6 requires 11.82 minutes to solve this problem. Figure 1 represents the corresponding schedule. In this schedule the server **L** loads first the job J_7 on the machine $M1$. This operation take one unit of time. Then the processing of this job begins from

Job	1	2	3	4	5	6	7	8
p_i	6	5	10	7	7	9	6	5
l_i	4	2	3	1	1	2	1	3
u_i	3	1	1	1	2	3	4	3

Table 1. Example instance for $n = 8$.

time $t = 1$ until $t = 7$. At this time the second server **U** is used to unload the J_7 from machine M1. This operation takes four units of time. The makespan in this schedule is equal to 46.

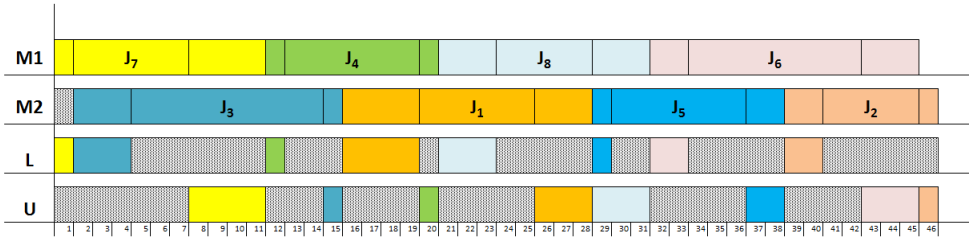

Fig. 1. The optimal schedule for the $n = 10$ problem instance

The problem is called *restrictive* because:

- once the job is loaded on the machine by the server **L**, it must be processed immediately by the machine.
- once the job is processed by the machine, it must be unloaded immediately by the server **U**.

If one of these conditions is not met, then the value of the optimal solution can be improved. In this case, a non-restrictive version of the problem occurs.

3 Mathematical Formulation of the Restrictive Model

In this case, the unloading of a job is carried out immediately after the end of its processing.

Notations:

- n the number of jobs
- $M = \{1, 2, ..., m\}$ set of the machines
- p_i the processing time of job i
- l_i the loading time of job i

- u_i the unloading time of job i
- B A large positive integer
- $\Omega_1 = \{1, 2, ..., n\}$ set of jobs to be processed on the machines
- $\Omega_2 = \{n+1, ..., 2n\}$ set of jobs to be processed on the loading server, each job i has a duration l_i
- $\Omega_3 = \{2n+1, ..., 3n\}$ set of jobs to be processed on the unloading server, each job i has a duration u_i
- $\Omega = \{1, 2, ..., 3n\}$ set of all the jobs

For the needs of modeling, we adopt the following notations: the ρ parameter will represent the duration of jobs, whether on the machine or on the servers. Thus:

$$\forall i \in \Omega_1 \quad \rho_i = p_i + l_i + u_i$$
$$\forall i \in \Omega_2 \quad \rho_i = l_{i-n}$$
$$\forall i \in \Omega_3 \quad \rho_i = u_{i-2n}$$

Variables:
C_i : the completion time of the job i
$x_{ik} = 1$ if job $i \in \Omega_1$ is processed on machine k and 0 otherwise
$z_{ij} = 1$ if job i is processed before job j and 0 otherwise.

$$min\ C_{max} \tag{1}$$
$$s.t.\ C_{max} \geq C_i, \quad \forall i \in \Omega_1, \tag{2}$$
$$\sum_{k=1}^{m} x_{ik} = 1, \quad \forall i \in \Omega_1, \tag{3}$$
$$C_i \geq \rho_i, \quad \forall i \in J \tag{4}$$
$$C_i \leq C_j - \rho_j + B(3 - x_{ik} - x_{jk} - z_{ij}), \qquad \forall i \neq j \in \Omega_1, k \in M \tag{5}$$
$$C_i \leq C_j - \rho_j + B(1 - z_{ij}), \quad \forall i \neq j \in \Omega_2 \tag{6}$$
$$C_i \leq C_j - \rho_j + B(1 - z_{ij}), \quad \forall i \neq j \in \Omega_3 \tag{7}$$
$$z_{ij} + z_{ji} = 1, \quad \forall i \neq j \in \Omega \tag{8}$$
$$C_i - (p_i + u_i) = C_{i+n}, \quad \forall i \in \Omega_1 \tag{9}$$
$$C_i = C_{i+2n}, \quad \forall i \in \Omega_1 \tag{10}$$
$$x_{ij}, z_{ik} \in \{0, 1\}, \qquad \forall i, j \in \Omega,\ j > i \tag{11}$$

In this MIP model we aim to minimize the makespan C_{max} (1). Constraints set (2) states that makespan of an optimal schedule is greater than or equal to the completion time of all executed jobs. In turns, the completion time of each job is at least greater than or equal to the sum of the loading, the unloading and the processing times of this job (4). Constraints (3) state that each job must be processed on exactly one machine. Constraints sets (5), (6) and (7) guarantee, respectively, that all jobs are scheduled on the loader, on the machines, and on

the unloader without overlapping. The next constraints (8) impose that for each couple of jobs (i, j), one must be processed before the other. Next, constraints (9) are used to calculate the completion time of each job i. Since we are dealing with the restrictive case, which states each job is immediately unloaded from the machine after its execution, then the completion time of the job, C_i, is equal to the completion time of the loading operation, C_{i+n}, plus the processing time and the unloading time $p_i + u_i$. Finally, constraints (10) are added to state that the completion time of the job i on the machine is equal to the completion time of unloading operation of the same job. Constraint sets (11) define variables $x_{i,j}$ and $z_{i,k}$ as binaries.

4 Variable Neighborhood Search

4.1 General Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a metaheuristic method based on systematic changes in the neighborhood structure initially proposed by Mladenović and Hansen [14]. The simplicity and efficiency of the VNS method has attracted several researchers the last decades and has lead to a large number of successful applications in a wide range of areas [3, 16].

In this paper, we use the General VNS (GVNS) variant to solve the problem in hand. GVNS employs the Variable Neighborhood Descent (VND) which consists of a powerful local search step in each neighborhood rather than a simple local search step in only one neighborhood per iteration. Thus, the VND method constitutes the intensification part of the VNS and it is analytically described in [5]. The pseudo-code of the algorithm is presented below (Algorithm 1).

Algorithm 1: GVNS

```
Data: x, l_max, k_max, t_max
Result: Solution x
Generate an initial solution x;
repeat
    l ← 1 ;
    repeat
        x' ← Shake(x, k_max, l_max);
        x'' ← VND(x', l_max);
        x, l ← NeighborhoodChange(x,x'',l);
    until l = l_max;
    t ← CpuTime() ;
until t > t_max;
```

The method **NeighborhoodChange**(x, x'', l) is used to change (or not) the current neighborhood structure. If the local optimum x'' is better than the incumbent x, then **NeighborhoodChange** keeps this solution instead of x (i.e.

$x \leftarrow x''$), and the search returns to $\mathcal{N}_1$; otherwise, it sets $l \leftarrow l + 1$ in order to find, if possible, a better solution in a different neighborhood.

VND (Algorithm 2) starts with an initial solution x_0 and continuously tries to construct a new improved solution from the current solution x by exploring its neighborhood $\mathcal{N}_l(x)$. The process continues to generate neighboring solutions until no further improvement can be made. In our implementation, we use the first improvement search strategy as we choose a random solution as the initial solution [9].

Algorithm 2: VND method

```
Data: x, l_max
Result: Solution x
repeat
    l ← 1;
    repeat
        Select x' ∈ N_l(x) such that f(x') < f(x);
        NeighborhoodChange(x,x',l);
    until l = l_max;
until no improvement is made;
return x
```

The aim of a Shaking procedure used within a VNS algorithm is to escape from local minima traps; thus, the Shaking method constitutes the diversification part of the VNS. The Shaking procedure performs a number of random jumps ($k \in \{1, 2, \ldots, k_{max}\}$) in a neighborhood $\mathcal{N}_l(x)$, $l \in \{1, 2, \ldots, l_{max}\}$ from a predefined set of neighborhoods. Note that, the l index is given as input by Algorithm 1. In this work, k_{max} was set equal to five based on some preliminary experiments.

Algorithm 3: Shake method

```
Data: x, k_max, l_max
Result: Solution x
for k = 1 to k_max do
    Select randomly x' ∈ N_l(x);
    x ← x';
end
return x
```

VNS uses a finite set of neighborhood structures denoted as $\mathcal{N}_l$, where $l \in \{1, 2, \ldots, l_{max}\}$. The l^{th} neighborhood of solution x, $\mathcal{N}_l(x)$, is a subset of the search space, which is obtained from the solution x by small changes. The VNS (Algorithm 1) includes an improvement phase in which a VND method is ap-

plied and a shaking phase used to escape local minima traps. These procedures are executed alternately until fulfilling a predefined stopping criterion. The stopping criterion of the proposed solution methodology was a maximum CPU time allowed for the VNS, equal to five minutes.

4.2 Neighborhood structures

To design an efficient VNS algorithm one must carefully select the neighborhoods structure to use. Some authors recommend the use of less than three neighborhood structures [8]. We have developed the following three neighborhood structures ($l_{max} = 3$) for the computational experiments:

- Neighborhood $\mathcal{N}_1(x) = Swap(x)$: The neighborhood set consists of all permutations that can be obtained by swapping two adjacent jobs in the solution x.
- Neighborhood $\mathcal{N}_2(x) = Swap2(x)$: It consists of all solutions obtained from the solution x swapping two random jobs.
- Neighborhood $\mathcal{N}_3(x) = Reverse(x)$: Given two jobs j and k we reverse the order of jobs being between those two jobs.

4.3 Initial solution

The initial solution is chosen as a random permutation of the jobs. From any sequence of the jobs, we can build the solution as follows: We start by loading the first job on the machine M_1, and the second one on machine M_2. In this case, the second job is directly loaded after the end of loading job 1 (i.e., without idle time). For each one of the following jobs j, as soon as the server **L** becomes available, we can load job j on one of the two available machines for processing. Otherwise, one should wait for one of the machines to be available before loading this job. It should be noted that, each time it must be checked that the end date of the unloading of job j does not overlap with another job which is being unloaded. If it is the case, it is necessary to shift the starting time of loading job j adequately.

4.4 Evaluation Function

Consider a permutation of jobs $\sigma = \{1, 2, \ldots, n\}$. To evaluate the value of the solution corresponding to σ we will proceed as follows. At $t = 0$ all resources are available. The job 1 is scheduled on machine 1. This means that the job is loaded on **L** which will take l_1 units of time. The machine $M1$ that is busy up to this point will start the processing of this job. At the end of this operation, the jobs are unloaded immediately from the machine using the resource **U**). Then we will schedule job 2 as soon as possible on machine 2. We may face two possibilities here. The first case is i) we will start loading this job on **L** just after the end of loading operation of job 1. In this case, the resource **U** is available when job 2 is to be unloaded (i.e. job 1 has finished unloading). The second case is ii) we

will shift the postpone the loading of job 2 so that at the time of unloading the resource **U** will be available. For the following jobs, we must choose the earliest starting date on the server and on one of the two machines so as to have an execution without idle time and the smallest completion time possible on the resource **U**. Finally, the value of the solution σ will be the completion time of the last scheduled job.

5 Computational results

We generated the data as suggested by Hasani et al. [10]. Hence, we randomly generated server load η in the interval $\{0.5, 1, 2\}$ for each server, where $\eta = E(s_i)/E(p_i)$ and $E(x)$ denotes the mean of x, and s_i can either represents the loading time l_i for the server **L** or the unloading time u_i for the server **U**. The processing times p_j were uniformly distributed in the interval (0, 100), and the loading and unloading times, respectively l_j and u_j were uniformly distributed in the interval $(0, 100\eta)$. Furthermore, we generated instances for $n \in \{15, 20\}$. Ten instances were randomly generated for each of the above values of η and for the additional values of n.

All tests presented in this section were conducted on a personal computer running Windows 7 with an Intel®Core(™) i7 vPro with a clock speed at 2.90 GHz CPU and 16 GB of RAM. Also, IBM CPLEX Optimizer v12.6 was used for the solution of the MIP optimization problems.

In Table 2, which is subdivided into two parts, we have reported the results of 60 instances solved by VNS and by CPLEX. These cases relate to problems of size $n = 15$ jobs and problems of size $n = 20$ jobs. For $n = 15$, the first column represents the instance k. The second represents the values of the server load η. The third column represents the best value found by VNS in a time limit of five minutes. The fourth column represents the best value (upper bound) found by CPLEX in one hour. Finally, the last column, represents the relative MIP gap (difference between the lower and upper bounds) computed by CPLEX.

In Table 2, for instances with 15 jobs, VNS finds a better solution than CPLEX in 87% of the cases. VNS performance is even better for large instances. In fact, for $n = 20$, VNS always finds a better solution for each case than CPLEX, whether the time limit is five minutes or five seconds. We report here only the solutions found in five minutes since they were better than those found in five seconds. Note that in both cases, we have written in bold the best values found by VNS. Finally, we should highlight the fact that, the MIP model for $n = 20$, $m = 2$, $\eta = 0.5$ was not able to solve optimally the first instance even after six hours.

$n = 15$					$n = 20$				
k	η	GVNS	CPLEX	GAP (%)	k	η	GVNS	CPLEX	GAP (%)
1	0.5	**818**	818	45.35	1	0.5	**1055**	2342	88.95
2	0.5	**873**	873	39.09	2	0.5	**1156**	2450	89.14
3	0.5	784	782	43.48	3	0.5	**991**	2466	89.45
4	0.5	**845**	845	39.43	4	0.5	**1151**	2496	89.51
5	0.5	**785**	785	33.20	5	0.5	**1052**	2616	89.77
6	0.5	**816**	816	34.80	6	0.5	**987**	2399	88.22
7	0.5	697	695	21.57	7	0.5	**1140**	2631	90.07
8	0.5	**832**	832	39.06	8	0.5	**989**	2755	88.85
9	0.5	**724**	724	35.90	9	0.5	**1033**	2522	88.90
10	0.5	**723**	723	29.05	10	0.5	**957**	2756	90.28
1	1	**1053**	1053	33.20	1	1	**1486**	3897	89.09
2	1	1178	1177	41.83	2	1	**1643**	3801	87.57
3	1	**1063**	1064	28.20	3	1	**1646**	3548	87.78
4	1	**1075**	1079	32.43	4	1	**1526**	4090	90.56
5	1	**1197**	1198	39.65	5	1	**1456**	3774	88.45
6	1	**1007**	1008	31.80	6	1	**1601**	3968	88.82
7	1	**1280**	1283	42.18	7	1	**1377**	4227	89.70
8	1	**1394**	1395	45.96	8	1	**1660**	3560	88.07
9	1	1171	1168	36.30	9	1	**1448**	4041	89.58
10	1	**1324**	1325	40.24	10	1	**1336**	4207	89.46
1	2	**2215**	2224	39.53	1	2	**2610**	6323	88.40
2	2	**2042**	2049	36.12	2	2	**2959**	6149	87.75
3	2	**2003**	2009	37.36	3	2	**2323**	7413	88.17
4	2	**1920**	1926	34.74	4	2	**2828**	6366	86.30
5	2	**1771**	1777	33.65	5	2	**2465**	6605	88.49
6	2	**1786**	1786	41.71	6	2	**2270**	6450	88.62
7	2	**1908**	1914	33.52	7	2	**2373**	6191	87.70
8	2	**2058**	2059	44.52	8	2	**2839**	6368	88.08
9	2	**2027**	2040	39.69	9	2	**2520**	6936	90.58
10	2	**2206**	2210	40.32	10	2	**2727**	7182	89.23

Table 2. Computational results

6 Conclusions and Future Work

This work studied the parallel machine scheduling problem with two servers in the restrictive case. Also, a mixed integer linear programming model and a variable neighborhood search approach were presented for the solution of the proposed problem. The VNS solution approach showed a very good computational performance and computed solutions of better quality than CPLEX for instances with 15 or 20 jobs. Studying problems with more than two machines consists an interesting research idea for future work. Also, a similar model corresponding to the non-restrictive case is also interesting as a generalization of the proposed model.

References

1. Abdekhodaee, A.H., Wirth, A.: Scheduling parallel machines with a single server: some solvable cases and heuristics. Computers & Operations Research **29**(3), 295–315 (2002)
2. Bektur, G., Saraç, T.: A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. Computers & Operations Research **103**, 46–63 (2019)
3. Benmansour, R., Sifaleras, A., Mladenović, N. (eds.): Variable Neighborhood Search. 7th International Conference, ICVNS 2019, Rabat, Morocco, October 3-5, 2019, Revised Selected Papers, LNCS, vol. 12010. Springer, Cham (2020)
4. Brucker, P., Dhaenens-Flipo, C., Knust, S., Kravchenko, S.A., Werner, F.: Complexity results for parallel machine problems with a single server. Journal of Scheduling **5**(6), 429–457 (2002)
5. Duarte, A., Mladenović, N., Sánchez-Oro, J., Todosijević, R.: Variable Neighborhood Descent, pp. 1–27. Springer International Publishing, Cham (2016)
6. Elidrissi, A., Benbrahim, M., Benmansour, R., Duvivier, D.: Variable neighborhood search for identical parallel machine scheduling problem with a single server. In: Variable Neighborhood Search. ICVNS 2019. Lecture Notes in Computer Science. vol. 12010, pp. 112–125. Springer (2019)
7. Elidrissi, A., Benmansour, R., Benbrahim, M., Duvivier, D.: Mip formulations for identical parallel machine scheduling problem with single server. In: 2018 4th International Conference on Optimization and Applications (ICOA). pp. 1–6. IEEE (2018)
8. Glover, F.W., Kochenberger, G.A.: Handbook of metaheuristics, vol. 57. Springer Science & Business Media (2006)
9. Hansen, P., Mladenović, N., Pérez, J.A.M.: Variable neighbourhood search: methods and applications. Annals of Operations Research **175**(1), 367–407 (2010)
10. Hasani, K., Kravchenko, S.A., Werner, F.: Block models for scheduling jobs on two parallel machines with a single server. Computers & Operations Research **41**, 94–97 (2014)
11. Jiang, Y., Zhang, Q., Hu, J., Dong, J., Ji, M.: Single-server parallel-machine scheduling with loading and unloading times. Journal of Combinatorial Optimization **30**(2), 201–213 (2015)

12. Kim, M.Y., Lee, Y.H.: Mip models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server. Computers & Operations Research **39**(11), 2457–2468 (2012)
13. Liu, G.S., Li, J.J., Yang, H.D., Huang, G.Q.: Approximate and branch-and-bound algorithms for the parallel machine scheduling problem with a single server. Journal of the Operational Research Society **70**(9), 1554–1570 (2019)
14. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & Operations research **24**(11), 1097–1100 (1997)
15. Ou, J., Qi, X., Lee, C.Y.: Parallel machine scheduling with multiple unloading servers. Journal of Scheduling **13**(3), 213–226 (2010)
16. Sifaleras, A., Salhi, S., Brimberg, J. (eds.): Variable Neighborhood Search. 6th International Conference, ICVNS 2018, Sithonia, Greece, October 4-7, 2018, Revised Selected Papers, LNCS, vol. 11328. Springer, Cham (2019)
17. Torjai, L., Kruzslicz, F.: Mixed integer programming formulations for the biomass truck scheduling problem. Central European Journal of Operations Research **24**(3), 731–745 (2016)
18. Werner, F., Kravchenko, S.A.: Scheduling with multiple servers. Automation and Remote Control **71**(10), 2109–2121 (2010)