Management and Production Engineering Review
Volume 12 • Number 3 • September 2021 • pp. 15–24
DOI: 10.24425/mper.2021.138527

The Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times

Anita Agárdi, Károly Nehéz

Institute of Information Science, University of Miskolc, Hungary

Received: 26 June 2020
Accepted: 16 August 2021

Abstract
Unrelated Parallel Machines Scheduling Problem (U-PMSP) is a category of discrete optimization problems in which various manufacturing jobs are assigned to identical parallel machines at particular times. In this paper, a specific production scheduling task the U-PMSP with Machine and Job Dependent Setup Times, Availability Constraint, Time Windows and Maintenance Times is introduced. Machines with different capacity limits and maintenance times are available to perform the tasks. After that our problem, the U-PMSP with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times is detailed. After that, the applied optimization algorithm and their operators are introduced. The proposed algorithm is the genetic algorithm (GA), and proposed operators are the order crossover, partially matched crossover, cycle crossover and the 2-opt as a mutation operator. Then we prove the efficiency of our algorithm with test results. We also prove the efficiency of the algorithm on our own data set and benchmark data set. The authors conclude that this GA is effective for solving high complexity parallel machine problems.

Keywords
production scheduling, parallel machines, setup times, time windows, genetic algorithm.

Introduction

The natural goal of manufacturing companies is to maximize profits. Maximizing profit may includes cost reduction. One of the main costs for production companies is the loss of production. Such production lost is the setup times of the machines i.e. switching from one job to another. In this paper a specific problem, the Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times is presented. In the case of our problem there are m machines and n jobs. Each job has a time window and processing time. Machines have production capacities, maintenance times (in this interval, the machine cannot make any jobs). The jobs also have setup times. Setup times depend on the last created jobs of the machines. The objective function is the minimization of the setup times. In this paper first, the Parallel Machines Scheduling Problem and their variants are introduced, then our problem, the Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times. After that, the Genetic Algorithm and the crossover and mutation techniques is presented. The genetic algorithm (GA) is a population-based metaheuristic algorithm. GA starts from a population of solutions and performs crossover and mutation operations on them, so the algorithm creates a new population. The algorithm performs the creation of the new population up to the termination condition. The efficiency of the proposed algorithm was also verified with different test data (benchmark and own). After the test result section conclusion remarks are presented.

Parallel Machines Scheduling Problem

In this section, the Parallel Machines Scheduling Problem is introduced. Each job can be assigned to one machine. There is a pre-defined setup time for

***Corresponding author**: Anita Agárdi – Institute of Information Science, University of Miskolc, Miskolc, Hungary, 3515, phone: +36 465 65 111/17-56, e-mail: agardianita@iit.uni-miskolc.hu*

© 2021 The Author(s). This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

each job. The problem is to solve which jobs are assigned to which machines, therefore the objective function is to minimize the setup times.

Many types and constraints of the task have been evolved over the years, which is illustrated in Fig. 1 in the components section. Additionally, machines may have some capacity limit in connection with production. This type of problem is called Capacitated Parallel Machines Scheduling (Lee & Liman, 1993). We may not do all the jobs, but the goal is to do all jobs in as little production time as possible, so the objective function is the minimization the sum of job completion time (Lee & Liman, 1993). The machines can be same, or unrelated (different capacity, setup times, different production cost etc.) (Lenstra et al., 1990). The maintenance in connection with production scheduling can be also considered (Lee & Chen, 2000). Machines can also be shut down in a predefined time, this factor can also be important (Hwang & Chang, 1998). The release date (Gharbi & Haouari, 2005) means, that a job should be started until a certain date. Delivery (Gharbi & Haouari, 2005) time is also important, this means, that a job must be done until a predefined time. The combination of release time and due date means the time window constraint. Hierarchical parallel machine scheduling means, that a job can be scheduled on a machine only when its hierarchy level is not higher than that of the machine (Zhang et al., 2009). Parallel Machine Scheduling Problem with server is a problem, where the setup is performed by a server (Kravchenko & Werner, 1997). It can be a single server problem, in which case a single server is included, or multiple server problem, where multiple servers are involved in the model. Job shop scheduling is also a well-known production optimization task, where a set of m machines and n jobs are given. Job i consists of a chain of m_i operations from set $O = \{1, \ldots, N\}$, each operation $i \in O$ belongs to job J_i and has to be processed on machine μ_i. The constraints are the followings: no one job is pre-empted, no two jobs are processed at the same time on the same machine, the maximum of the completion times (and makespan) is minimized (Dell'Amico & Trubian, 1993). Flexible Job Shop Scheduling Problem is a generalization of the classical Job Shop Scheduling Problem, where operations can be processed on any among a set of available machines (Pezzella et al., 2008). The preemption (job splitting) can also be allowed (Graham et al., 1979). It can occur, that some job has precedence (Graham et al., 1979). The objective function can be the minimization of the setup times, which is a loss in production, while at this time, the production cannot be done on a machine (Damodaran et al., 2009). Another objective function can be the minimization of the total tardiness (Liaw et al., 2003). This means that we allow work outside of the due date, but there is a penalty point for being late. Average tardiness means the average difference between the completion time and the due date of a single job

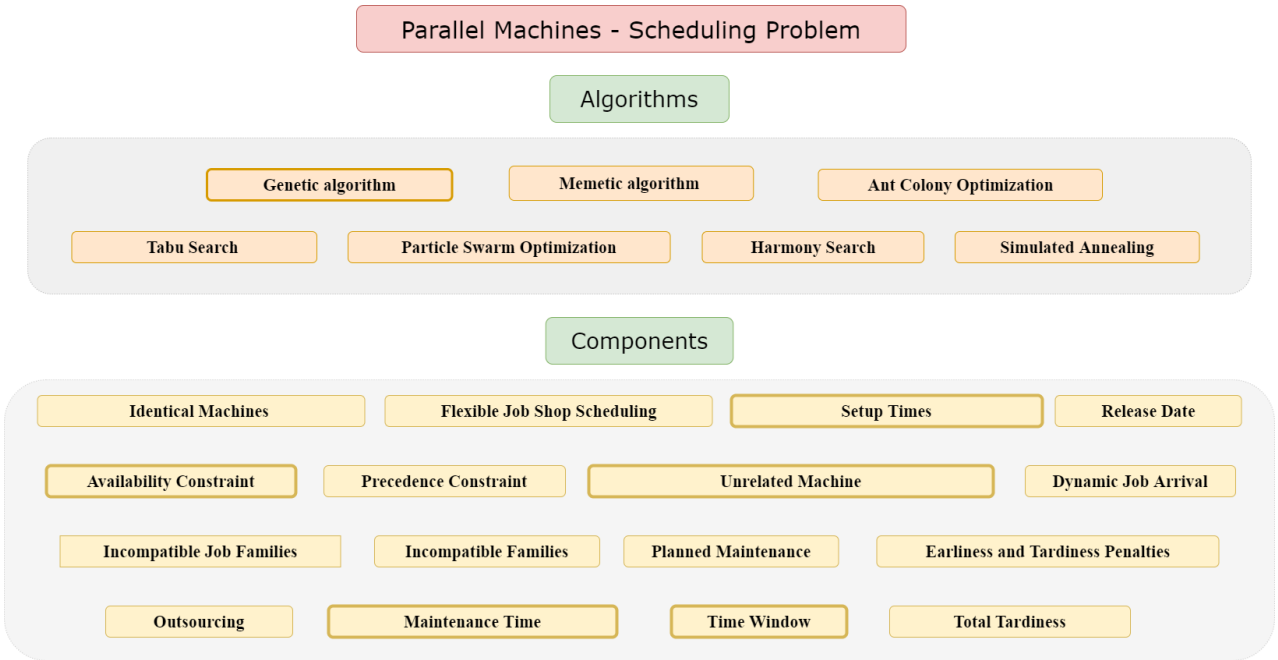

Fig. 1. The applications of the metaheuristics in Parallel Machines Scheduling Problem (PMSP)

(Chaudhry & Khan, 2016). In the case of the minimization of the total weighted tardiness some jobs are more important than others (Chaudhry & Khan, 2016). The number of tardy jobs can also be minimized (Chaudhry & Khan, 2016). Mean completion time means that the average time taken to finish a single time can also be considered (Chaudhry & Khan, 2016). Maximum flow time is an objective function when the longest time a job spends in the shop and schedule's cost is directly related to its longest job (Chaudhry & Khan, 2016). The Mean flow time means the average time a single job spends in the shop and a schedule's cost is directly related to the average time it takes to process a single job (Chaudhry & Khan, 2016). The total workload of machines can also be considered in the objective function (Chaudhry & Khan, 2016).

In the following, some publications have been presented, that address the unrelated parallel machines scheduling problem.

Li et al. (2015) investigated the unrelated parallel machine scheduling problem with energy and tardiness cost, where the objective is to minimize the penalty cost of tardy jobs and energy consumption cost of machines. The energy consumption cost of machines consists of three parts: running energy consumption cost, waiting energy consumption cost, and the warmup energy consumption cost.

Vallada & Ruiz (2011) used genetic algorithm to solve the UPMSP with sequence dependent setup time. They used a representation mode where each machine has a string that stores the jobs that belong to the machine. Genetic algorithm was also combined with the use of specific local search.

Lin et al. (2011) investigated the minimization of regular performance measures in unrelated parallel machine scheduling problems. A single string representation was used for the task, where a separator character (*) indicates which job will belong to which machine. The authors solved the problem with genetic algorithm.

Yang et al. (2012) presented the task of UPMSP with aging effects and multi-maintenance activities. In this problem the machine can have several maintenance activities over the scheduling horizon, and the aging of the machines is also considered.

The Parallel Machine Scheduling Problems are generally NP-hard problems. Many metaheuristics have already solved the problem. Fig. 1 shows how many different algorithms the problem has already been solved and how many components it has. Our task is marked with bold letter and with a thicker frame. We included 7 metaheuristics in our search and divided the results into 16 components.

We used the genetic algorithm to solve our problem because it is a frequently used method in parallel machine scheduling. In the following are some of these articles are introduced.

Woo et al. (2017) investigated the unrelated parallel machine scheduling problem with time dependent deterioration and multiple rate-modifying activities using the genetic algorithm. Different genetic algorithms compared representations such as GA_TS (GA with triple-dimensional string) GA_RD (GA with completion time rule-based dispatching heuristic), and GA_RD proved to be much better. For GA_TS, the representation consists of three parts, a job order, an assignment part, and a machine index part. While in the case of GA_RD, the representation consists of a single part, a job sequence.

Chung & Kim (2016) also applied a genetic algorithm to the machine scheduling problem with step-deteriorating jobs and rate-modifying activities problem. The authors introduced two types of chromosome representations that were compared based on efficiency. The GADS consists of a two-dimensional string array, which is a sequence array and an assignment array. The sequence array indicates the order of the jobs, and the assignment array indicates which job will belong to which machine. GASC is a one-dimensional string array where the job order is separated by delimiters (*). The delimiters determine which job will belong to which machine. For small data sets, GASC is also efficient, but for larger data sets, GADS proved to be much more efficient based on the test results.

Al-Shayea et al. (2020) investigated the Two Identical Parallel Machines solved the Subjected to Release Times, Delivery Times and Unavailability Constraints problem with a genetic algorithm. A permutation representation mode was proposed, the representation mode consisted of a single string that determines the order of the jobs. In this way of representation, the operators of each genetic algorithm (crossing, mutation) can be easily implemented.

Tavakkoli-Moghaddam et al. (2009) investigated the unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints with genetic algorithm. A single-string permutation representation mode was applied to the task. They also used the technique where the string contains the serial number of the jobs and a separator character (*). Here again, the separator character determines which job will belong to which machine.

Rajakumar et al. (2006) detailed workflow balancing in parallel machines with genetic algorithm. Genetic algorithm was compared with construction heuristics. Three construction strategies were used,

which are the followings: random, shortest processing time, longest processing time. According to the random method, the job-machine assignment is performed randomly. According to the shortest processing time, jobs are sorted in ascending order by processing time and assigned to machines. Longest processing time is the inverse of shortest processing time. Based on the test results, genetic algorithm proved to be the best, but the longest processing time did not lag far behind in terms of fitness value either. Random and shortest processing time approaches were not effective.

The Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times

In this section, Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times is presented. In our case, the number of jobs and machines and the capacity constraints are known in advance. Capacity constraint means, how long a machine can run. Jobs have processing time, which may by machine. There is a setup time between each job, which also depends on the machine. Jobs also have time windows, which means, that jobs must be created within a certain time interval. The objective function is the minimization of the setup time. Every job has to be done within the time window.

In Fig. 2, the Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times is presented. Three machines make seventeen jobs. All machines have reset state from which after setup the first job can be done. After finishing the work, the machines must be reset. In Fig. 2, after 0 hour setup time the Machine 1 make Job 10, then 1 hour setup time the Job 13, then after 1 hour setup time Job 5, then after 1 hour setup time Job 8, then after 2 hour setup time Job 6, then after 1 hour setup time Job 12. There is also a maintenance time between Job 6 and Job 12. All processing time of the jobs is 21 hours. Machine 2 has 30 hours of processing time. Machine 2 makes Job 4, Job 17, Job 1, Job 14, Job 9, Job 15. There is also a maintenance time between Job 9 and Job 15. Machine 3 has 37 hours of processing time. This machine has done Job 11, Job 16, Job 2, Job 7, Job 3. There is also a maintenance time between Job 11 and Job 16.

In practice, the parallel machines task most often appears in manufacturing. Operations must be distributed among multiple machines. Operations have machining time and operations must be performed for deadline. The machines need to be maintained, so it is possible that production will stop for a while. Machines can have a cost (which is a one-time cost). The jobs can have also cost (manufacturing cost). Parallel machines also appear in the project schedule. Then the machines actually mean the employees. Tasks should be distributed among employees, taking care of the working hours, rest periods, lunch breaks, etc. of the employees, as well as ensuring that the right employee will get the right project.

In the following, we present the mathematical model of the Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times.

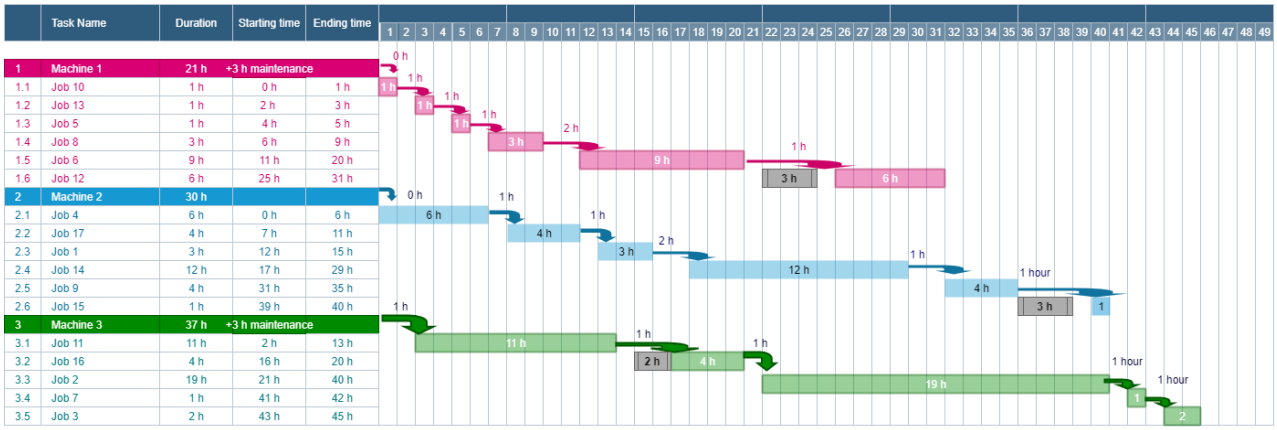

Fig. 2. Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times

Table 1
The symbols and their explanations

Symbol	Explanation of the symbol
n_{jobs}	Number of jobs
$JO = \{jo_i\}$	Jobs
$PT = \{pt_i\}$	Processing time of each job
n_{machines}	Number of machines
$MA = \{ma_i\}$	Machines
$C = \{c_i\}$	Production capacity of machines
$TW = \{tw_i\}$	The time windows of the jobs, where $tw_i = [a_i b_i]$
$ST = \{st_{i,j,k}\}$	Setup times of the jobs. The 0 idex is the reset state of the machine.
$MT = \{mt_{i,j}\}$	Maintenance times, where $mt_{i,j} = [mta_{i,j} mtb_{i,j}]$
$x^{ma}_{jo_1 jo_2}$	Decision variable. 1 if the job jo_1 and job jo_2 belongs to the machine ma, and after job jo_1 the machine performs job jo_2 0 else
y_{jo_i}	Starting time of job jo_i

Objective function: $\min Z$
where:

$$Z = \sum_{jo=1}^{n_{\text{jobs}}} \sum_{ma=1}^{n_{\text{machines}}} st_{0,jo,ma} x^{ma}_{0,jo} + \sum_{jo_1=1}^{n_{\text{jobs}}} \sum_{jo_2=1}^{n_{\text{jobs}}} \sum_{ma=1}^{n_{\text{machines}}} st_{jo_1,jo_2,ma} x^{ma}_{jo_1,jo_2} + \sum_{jo_1=1}^{n_{\text{jobs}}} \sum_{ma=1}^{n_{\text{machines}}} st_{jo_1,0,ma} x^{ma}_{jo_1,0}. \quad (1)$$

Constraint 1: Each job must be performed once:

$$\sum_{jo_1=1}^{n_{\text{jobs}}} \sum_{ma=1}^{n_{\text{machines}}} x^{ma}_{jo_1,jo_2} = 1 \; \forall jo_2 \in JO, \quad (2)$$

$$\sum_{jo_2=1}^{n_{\text{jobs}}} \sum_{ma=1}^{n_{\text{machines}}} x^{ma}_{jo_1,jo_2} = 1 \; \forall jo_1 \in JO. \quad (3)$$

Constraint 2: Continuity of production:

$$\sum_{jo_1=0}^{n_{\text{jobs}}} x^{ma}_{jo_1,jo_2} = \sum_{jo_2=0}^{n_{\text{jobs}}} x^{ma}_{jo_1,jo_2} \; \forall ma \in MA. \quad (4)$$

Constraint 3: Time window:

$$a_{jo_1} \leq y^{ma}_{jo_1} \; \forall ma \in MA, \; \forall jo_1 \in JO, \quad (5)$$

$$y^{ma}_{jo_1} + st_{jo_1,jo_2,ma} \leq b_{jo_1} \; \forall ma \in MA, \; \forall jo_1 \in JO. \quad (6)$$

Constraint 4: Machines do not exceed the availability constraint limit:

$$\sum_{jo_1=0}^{n_{\text{jobs}}} \sum_{jo_2=0}^{n_{\text{jobs}}} pt_{jo_2,ma} x^{ma}_{jo_1,jo_2} \leq c_m \; \forall ma \in MA. \quad (7)$$

Constraint 5: Maintenance times of the jobs must be taken into account:

$$y^{ma}_{jo_1} + st_{jo_1,jo_2,ma} \leq mta_{i,j}, \quad ma \in MA, \quad jo_1 \in JO, \quad mta_{i,j} \in mt_{i,j} \in MT \quad (8)$$

and

$$y^{m}_{jo_1} \geq mtb_{i,j} \; m \in M, \quad jo_1 \in JO, \quad mtb_{i,j} \in mt_{i,j} \in MT \quad (9)$$

in at least one case.

The optimization algorithm

Parallel machines problems are NP hard, heuristic algorithms are required to get a solution with acceptable accuracy within a reasonable amount of time. In addition, our problem is difficult, because it includes the following constraints: machine dependent setup times, job dependent setup times, availability constraints, multiple time windows and maintenance times of the machines.

The genetic algorithm is a classical algorithm that, as we have shown in the literature section, has already effectively solved the parallel machines scheduling problems in many cases.

The genetic algorithm (GA) is an evolution-inspired partial search algorithm. The algorithm encodes potential solutions of a specific problem. The algorithm begins with the population of the solutions (most often random solutions). Iteratively produces elements of the next population using elements of the previous population. Part of the elements of the previous population will be a part of the next population. The rest of the next population is filled with individuals created with crossover and mutation. The crossover produces generally from two parent individuals two children individuals. The mutation is a small change in an individual. The strategy of selecting individuals from the previous population is called selection.

The better everyone in the previous population is, the more likely that these individuals will be selected. The "goodness" of individuals is characterized by a fitness function. The better an individual is, the higher the value of fitness (Whitley, 1994).

BEGIN PROCEDURE
Step 1. Initialization the population
Step 2. Calculation the fitness values of each solution
WHILE (termination criterion is not met) ***DO***
Step 3. Bringing certain parents to the next generation unaltered (elitism)
Step 4. Recombination of selected parent pairs (crossover)
Step 5. Mutation of selected new individuals
Step 6. Evaluation of new individuals
Step 7. Selecting individuals of the next generation
END WHILE
END PROCEDURE

Fig. 3. Pseudo code of genetic algorithm

The first step is the initialization of the population. Then the fitness values of each individual are calculated. The next step is creation of the next population. In this step, the elitism is applied. Then the selected individuals are recombined. After that, some newly created individuals can also be mutated.

The most common representation of the genetic algorithm is binary and permutation representation. In this article permutation representation is used for the Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times.

Fig. 4 illustrates the representation of the problem. In the case of the Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times permutation representation is used. We chose the permutation representation because it is a simple, easy-to-implement structure. Classical genetic operators (mutation and crossing) can also be easily implemented, we do not need to use a new operator. In addition, it has been used by many authors in research on parallel machines, for example.

The element of the permutation represents each job. We assign jobs to machines by taking each job in sequence from the beginning of the permutation and assigning it to the same machine until any constraint is violated. If any constraint is violated, then we assign it to the next machine.

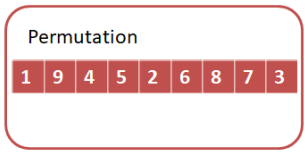

Fig. 4. Representation of a permutation

In the following the operators of the genetic algorithm will be presented.

Fig. 5 presents the Partially Matched Crossover (PMX) (Chan & Tansri, 1994). During this crossover technique first two parents are selected from the previous population. Then a fitting section is created. In the next step, the genes are paired. In our example, the pairs are the followings: $(5, 9)$, $(2, 7)$, $(6, 1)$, $(8, 8)$. These genes are swapped in both parents. This procedure results in the chromosomes of the children.

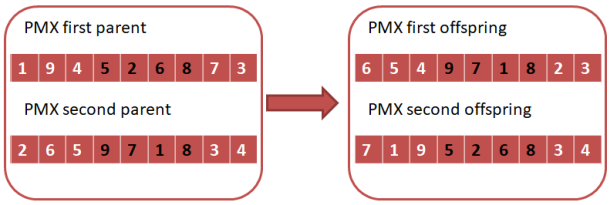

Fig. 5. Partially matched crossover

Fig. 6 illustrates the Order Crossover (OX) (Chan & Tansri, 1994). In the case of this crossover fitting sections are also created in the two selected parents. The elements of the fitting section of the first parent are deleted in the second parent, and the elements of the fitting section of the second parent are deleted in the first parent. The deleted locations are indicated in Fig. 6 with H letter. The H letters are pushed away in the fitting sections. The last step is the following: the first child gets the elements of the fitting section

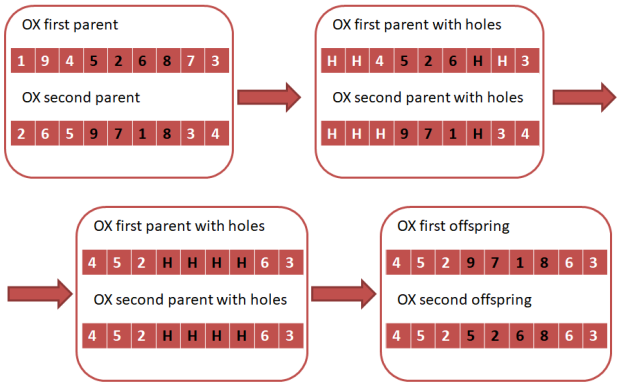

Fig. 6. Order crossover

of the second parent, and the second child gets the elements of the fitting section of the first parent.

Fig. 7 present Cycle crossover (CX) (Chan & Tansri, 1994). In the case of this crossover also two parents are selected from the previous population. In the case of this method looking for cycles. The first child gets the first gene of the first parent. The second child gets the first gene of the second parent. In our example the first gene of the first child will be 1, the first gene of the second parent will be 2. The next step is searching for 2 in the first parent. This gene is in the fifth position of the first parent. The fifth position of the second parent is 6, so in the fifth position of the first child will be 2, and 6 in the second child. This procedure is continued until the circle is closed, which is done in the example with $(6, 1)$ pair. After that, the remaining elements of the first parent will get the second child, and the remaining elements of the second parent will get the first child.

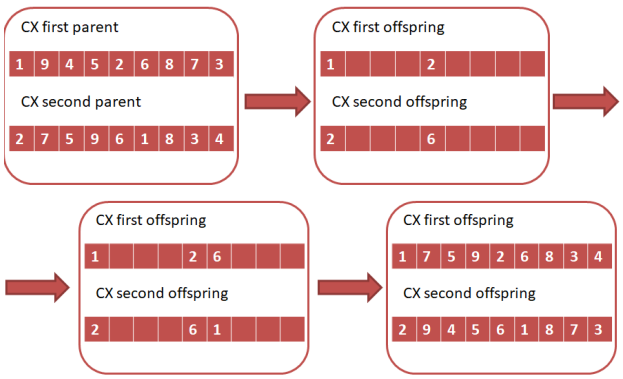

Fig. 7. The cycle crossover

Fig. 8 illustrates the applied mutation technique. In the case of the 2-opt mutation (Englert et al., 2007) a section is selected, and the elements of the section are exchanged. In our example the $(5, 8)$ and $(2, 6)$ elements are exchanged.

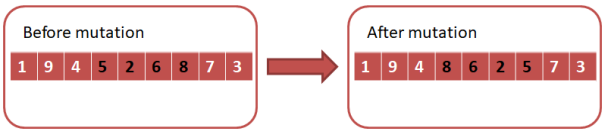

Fig. 8. The 2-opt mutation

Results and discussion

In this section, test results have been presented. Our algorithm is tested first with a benchmark dataset from (Tanaka, 2019). In this benchmark dataset the problem is the Total Tardiness Problem on Identical Parallel Machines, which is not as difficult task as ours. In the case of this benchmark problem also given the number of machines and tasks in advance. The tasks have also processing times. Each task has only due date, which means that the task must be completed to that date. The machines are identical, they do not have any maintenance times and capacity constraints. Jobs do not have any setup times. The objective function is the completion of all jobs with no due date. Table 2 illustrates some test results of the benchmark dataset. In the table, the results that achieved the best known result of the benchmark data set so far are shown in bold and italics. This was achieved in 4 of 7 cases.

Since we could not find a benchmark data set for our own task (Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraint, Time Windows and Maintenance Times), we created our own data set. In the following, we will present our own data set, followed by the test result. Table 3 indicates the parame-

Table 2
Test results on Tanaka benchmark dataset (Tanaka, 2019)

Benchmark dataset			Result	
Name of the dataset	Number of jobs	Number of machines	Number of created jobs (average)	Number of machines (average)
20_02_02_06_001	20	2	***20***	***2***
20_02_02_06_002	20	2	***20***	***2***
20_02_02_06_003	20	2	***20***	***2***
20_02_02_06_004	20	2	19	2
25_04_02_08_001	25	4	22	4
25_04_02_08_002	25	4	***25***	***4***
25_04_02_08_003	25	4	23	4

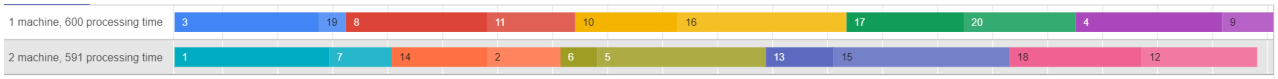

Fig. 9. Test result of the 20_02_02_06_001 dataset

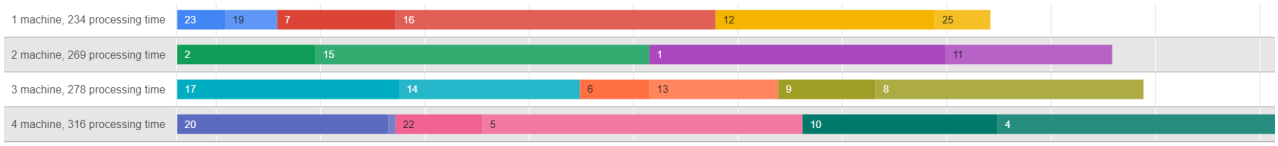

Fig. 10. Test result of the 25_04_02_08_001 dataset

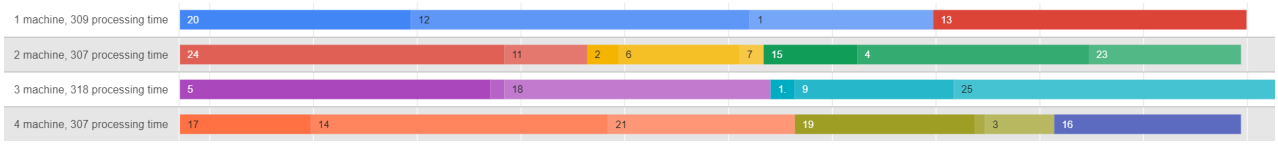

Fig. 11. Test result of the 25_04_02_08_002 dataset

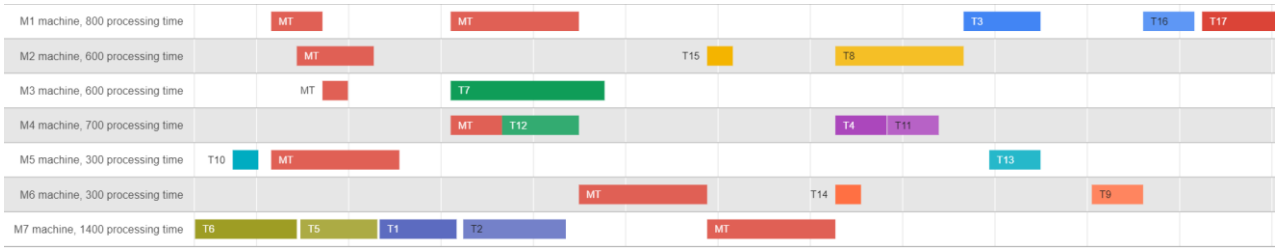

Fig. 12. Test result of our dataset

ters of the machines, Table 4 presents the parameters of the task. Specifying the setup parameter in our article would be lengthy as the values vary from machine to machine and from job to job. Values move in the interval [0, 50].

Table 3
Machine parameters

Machine	Capacity	Maintenance time
M1	5000	[300, 500], [1000, 1500]
M2	6000	[400, 700]
M3	7000	[500, 600]
M4	6000	[1000, 1200]
M5	8000	[300, 800]
M6	10000	[1500, 2000]
M7	9000	[2000, 2500]

Fig. 12 presents the result of our test data. The figure shows that all the job has been completed, no job is cancelled. Based on this result and benchmark results, our algorithm proved its effectiveness in solving the parallel machines scheduling problem.

Table 4
Task parameters

Task	Start time window	End time window	Processing time
T1	0	1200	300
T2	1200	3000	400
T3	3000	5000	300
T4	2500	4200	200
T5	1125	3600	300
T6	0	1000	400
T7	1000	2000	600
T8	2500	4500	500
T9	3500	4200	200
T10	150	2200	100
T11	2500	5000	200
T12	1800	1200	300
T13	3100	4200	200
T14	2500	3000	100
T15	2000	3500	100
T16	3700	4200	200
T17	3800	5000	300

Conclusions

Parallel Machines Scheduling is an important industrial problem because cost-effective production is one of the long-term goals of production companies. In this paper first, the variants of the Parallel Machines Scheduling has been detailed. After that, Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times problem has been introduced. Then Genetic Algorithm approach has been detailed because our problem is solved with this metaheuristic algorithm. The applied crossover and mutation operators have also been presented. We then verify the efficiency of our algorithm with our tests and benchmarks. In this paper, we demonstrated the efficiency of our genetic algorithm approach using partially matched crossover, cycle crossover and 2-opt operators for complex parallel machines scheduling tasks.

References

Al-Shayea, A.M., Saleh, M., Alatefi, M., and Ghaleb, M. (2020). *Scheduling Two Identical Parallel Machines Subjected to Release Times, Delivery Times and Unavailability Constraints*. Processes, 8, 9, 1025. DOI: 10.3390/pr8091025.

Chan, K.C. and Tansri, H. (1994). *A study of genetic crossover operations on the facilities layout problem*. Computers & Industrial Engineering, 26, 3, 537–550. DOI: 10.1016/0360-8352(94)90049-3.

Chaudhry, I.A. and Khan, A.A. (2016). *A research survey: review of flexible job shop scheduling techniques*. International Transactions in Operational Research, 23, 3, 551–591. DOI: 10.1111/itor.12199.

Chung, B.D. and Kim, B.S. (2016). *A hybrid genetic algorithm with two-stage dispatching heuristic for a machine scheduling problem with step-deteriorating jobs and rate-modifying activities*. Computers & Industrial Engineering, 98, 113–124. DOI: 10.1016/j.cie.2016.05.028.

Damodaran, P., Hirani, N.S., and Velez-Gallego, M.C. (2009). *Scheduling identical parallel batch processing machines to minimise makespan using genetic algorithms*. European Journal of Industrial Engineering, 3, 2, 187–206. DOI: 10.1504/ejie.2009.023605.

Dell'Amico, M. and Trubian, M. (1993). *Applying tabu search to the job-shop scheduling problem*. Annals of Operations research, 41, 3, 231–252. DOI: 10.1007/bf02023076.

Englert, M., Röglin, H., and Vöcking, B. (2007). *Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP*. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, 1295–1304. DOI: 10.1007/s00453-013-9801-4.

Gharbi, A. and Haouari, M. (2005). *Optimal parallel machines scheduling with availability constraints*. Discrete Applied Mathematics, 148, 1, 63–87. DOI: 10.1016/j.dam.2004.12.003.

Graham, R.L., Lawler, E.L., Lenstra, J.K., and Kan, A.R. (1979). *Optimization and approximation in deterministic sequencing and scheduling: a survey*. In Annals of discrete mathematics, 5, 287–326. DOI: 10.1016/s0167-5060(08)70356-x.

Hwang, H.C. and Chang, S.Y. (1998). *Parallel machines scheduling with machine shutdowns*. Computers & Mathematics with Applications, 36, 3, 21–31. DOI: 10.1016/s0898-1221(98)00126-6.

Kravchenko, S.A. and Werner, F. (1997). *Parallel machine scheduling problems with a single server*. Mathematical and Computer Modelling, 26, 12, 1–11. DOI: 10.1016/s0895-7177(97)00236-7.

Lee, C.Y. and Chen, Z.L. (2000). *Scheduling jobs and maintenance activities on parallel machines*. Naval Research Logistics (NRL), 47, 2, 145–165. DOI: 10.1002/(sici)1520-6750(200003)47:2%3C145::aid-nav5%3E3.0.co;2-3.

Lee, C.Y. and Liman, S.D. (1993). *Capacitated two-parallel machines scheduling to minimize sum of job completion times*. Discrete Applied Mathematics, 41, 3, 211–222. DOI: 10.1016/0166-218x(90)90055-h.

Lenstra, J.K., Shmoys, D.B., and Tardos, E. (1990). *Approximation algorithms for scheduling unrelated parallel machines*. Mathematical programming, 46, 1–3, 259–271. DOI: 10.1007/bf01585745.

Li, Z., Yang, H., Zhang, S., and Liu, G. (2015). *Unrelated parallel machine scheduling problem with energy and tardiness cost*. The International Journal of Advanced Manufacturing Technology, 84, 1–4, 213–226. DOI: 10.1007/s00170-015-7657-2.

Liaw, C.F., Lin, Y.K., Cheng, C.Y., and Chen, M. (2003). *Scheduling unrelated parallel machines to minimize total weighted tardiness*. Computers & Operations Research, 30, 12, 1777–1789. DOI: 10.1016/s0305-0548(02)00105-3.

Lin, Y.K., Pfund, M.E., and Fowler, J.W. (2011). *Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems*. Computers & Operations Research, 38, 6, 901–916. DOI: 10.1016/j.cor.2010.08.018.

Pezzella, F., Morganti, G. and Ciaschetti, G. (2008). *A genetic algorithm for the flexible job-shop scheduling problem*. Computers & Operations Research, 35, 10, 3202–3212. DOI: 10.1016/j.cor.2007.02.014.

Rajakumar, S., Arunachalam, V.P. and Selladurai, V. (2006). *Workflow balancing in parallel machines through genetic algorithm*. The International Journal of Advanced Manufacturing Technology, 33, 11–12, 1212–1221. DOI: 10.1007/s00170-006-0553-z.

Tanaka, S., *Total Tardiness Problem on Identical Parallel Machines*. https://sites.google.com/site/shunjitanaka/pmtt [Accessed: 2019.12.23.]

Tavakkoli-Moghaddam, R., Taheri, F., Bazzazi, M., Izadi, M. and Sassani, F. (2009). *Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints*. Computers & Operations Research, 36, 12, 3224–3230. DOI: 10.1016/j.cor.2009.02.012.

Vallada, E. and Ruiz, R. (2011). *A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times*. European Journal of Operational Research, 211, 3, 612–622. DOI: 10.1016/j.ejor.2011.01.011.

Whitley, D. (1994). *A genetic algorithm tutorial*. Statistics and computing, 4, 2, 65–85. DOI: 10.1007/bf00175354.

Woo, Y.-B., Jung, S., and Kim, B.S. (2017). *A rule-based genetic algorithm with an improvement heuristic for unrelated parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities*. Computers & Industrial Engineering, 109, 179–190. DOI: 10.1016/j.cie.2017.05.007.

Yang, D.-L., Cheng, T.C.E., Yang, S.-J., and Hsu, C.-J. (2012). *Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities*. Computers & Operations Research, 39, 7, 1458–1464. DOI: 10.1016/j.cor.2011.08.017.

Zhang, A., Jiang, Y., and Tan, Z. (2009). *Online parallel machines scheduling with two hierarchies*. Theoretical Computer Science, 410, 38–40, 3597–3605. DOI: 10.1016/j.tcs.2009.04.007.