J Intell Manuf (2018) 29:423–437
https://doi.org/10.1007/s10845-015-1117-6

Design of an efficient genetic algorithm for resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions

Mojtaba Afzalirad[1] · **Masoud Shafipour**[1]

Received: 4 November 2014 / Accepted: 14 June 2015 / Published online: 24 June 2015

Abstract This study addresses a resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions. Majority of the traditional scheduling problems in parallel machine environment deal with machine as the only resource. However, other resources such as labors, tools, jigs, fixtures, pallets, dies, and industrial robots are not only required for processing jobs but also are often restricted. Considering other resources makes the scheduling problems more realistic and practical to implement in manufacturing environments. First, an integer mathematical programming model with the objective of minimizing makespan is developed for this problem. Noteworthy, due to NP-hardness of the considered problem, application of meta-heuristic is avoidable. Furthermore, two new genetic algorithms including a pure genetic algorithm and a genetic algorithm along with a heuristic procedure are proposed to tackle this problem. With regard to the fact that appropriate design of the parameters has a significant effect on the performance of algorithms, hence, we calibrate the parameters of these algorithms by using the response surface method. The performance of the proposed algorithms is evaluated by a number of numerical examples. The computational results demonstrated that the proposed genetic algorithm is an effective and appropriate approach for our investigated problem.

Keywords Unrelated parallel machine scheduling · Resource constraints · Machine eligibility restriction · Genetic algorithm

✉ Mojtaba Afzalirad
Mojtaba.afzalirad@ustmb.ac.ir;
Mojtaba_afzalirad@yahoo.com

[1] Department of Industrial Engineering, Mazandaran University of Science and Technology, P.O. Box 734, Babol, Iran

Introduction

Scheduling is to simply assign a set of jobs to a set of machines with respect to operational constraints such that optimal usage of available resources is obtained. Because of the growing cost of raw material, labor, energy and transportation, scheduling plays an essential role in production planning of manufacturing systems, and it is one of the most important issues for survival in the modern competitive market place. Unrelated parallel machine scheduling problems appear in many manufacturing environments, such as in semiconductor manufacturing (Kim et al. 2002; Wang et al. 2013), the spinning of acrylic fibers for production in the textile industry (Silva and Magalhaes 2006), in steel making industry (Pan et al. 2013), in preheat treatment stage of automobile gear manufacturing process (Gokhale and Mathirajan 2012), block transportation in shipyards (Hu et al. 2010) and scheduling jobs in a printed wiring board manufacturing environment (Yu et al. 2002; Bilyk and Monch 2012).

Parallel machine scheduling problem has been one of the topics of interest for many researchers in the past few decades. Most of these studies considered the machines as only resource which is restricted. However, in real life manufacturing systems other resources such as machine operators and tools are constrained and it is illogical to consider that there are always enough resources for processing a job. It is clear that the resource-constrained parallel machine scheduling problem is more difficult than the simple parallel machine scheduling problem. In a resource constrained parallel machine scheduling problem (RCPMSP), an operation can be performed when the machine and other resources are available during the process of operation. In these problems, besides the schedule of jobs on machines, the schedule of jobs on other resources and their interactions should be considered. In RCPMSP for a specific sequence of jobs on machines, there is an optimal sequence of jobs on resources and conversely, for a specific sequence of jobs on resources, there is an optimal sequence of jobs on machines (Mehravaran and Logendran 2013). On the other hand, jobs may often not be processed on any of the available machines but rather must be processed on a machine belonging to a specific subset of the machines (Pinedo 1995). This constraint which is called under different names, such as machine eligibility restrictions, processing set restrictions and scheduling typed task systems is also widely encountered in real scheduling environments.

There are some works done related to the unrelated or identical parallel machine scheduling problem with machine eligibility restrictions and other operational constraints. Centeno and Armacost (1997) investigated the identical parallel machine scheduling problem (PMSP) with release dates and machine eligibility restrictions to minimize the maximum lateness. They developed a heuristic algorithm that resulted from combining the least flexible job first rule (LFJ) and the least flexible machine first rule (LFM). Later, Centeno and Armacost (2004) developed a heuristic algorithm that integrated LFJ, LFM and the longest processing time (LPT) for the aforementioned problem to minimize makespan. Liao and Sheen (2008) considered the identical PMSP with machine availability and eligibility constraints while minimizing the makespan. Sheen et al. (2008) developed a branch and bound algorithm applying several immediate selection rules for solving the identical PMSP with machine availability and eligibility constraints while minimizing the maximum lateness. Gokhale and Mathirajan (2012) addressed identical parallel machine scheduling problem with release dates, machine eligibility restrictions and sequence dependent setup times to minimize the total weighted flow time. They presented a mathematical model and a few heuristic algorithms to solve the considered problem. A mixed-integer programming model and some local search heuristics have been proposed by Wang et al. (2013), for unrelated PMSP incorporating release dates, machine eligibility restrictions and sequence dependent setup times. As indicated above, none of these researchers taken into account the resource constraints. Only Edis and Ozkarahan (2012) considered machine eligibility and resource constraints for identical PMSP simultaneously, however, they did not propose any meta-heuristic algorithm for this problem. To the best of our knowledge, this research is the first to propose a meta-heuristic algorithm for the mentioned problem.

The first studies considered the use of meta-heuristics for the PMSP have been conducted by Cheng et al. (1995) and Cheng and Gen (1997). Recently, Joo and Kim (2012) proposed two meta-heuristics, genetic algorithm (GA) and a new population-based evolutionary meta-heuristic called self-evolution algorithm (SEA) for unrelated PMSP with sequence dependent setup times and ready times. Arnaout et al. (2014) presented a two-stage ant colony optimization algorithm to minimize the makespan on unrelated PMSP with sequence dependent setup times.

On the other hand, there are only few papers that have considered the resource constraints in their scheduling problems. Ventura and Kim (2000) addressed identical PMSP with additional resource constraints and developed an integer programming model to minimize the total absolute deviation of job completion times about common due date. They proved that the problem can be solved in polynomial time when there exists one single type of additional resource and the resource requirements per job are zero or one. Later, Ventura and Kim (2003) considered an identical PMSP with unit processing times, non-common due dates and additional resource constraints. They developed an integer programming model and used lagrangian relaxation approach to obtain tight lower bounds and near-optimal solutions. Chen (2005) proposed a heuristic method to minimize makespan in unrelated PMSP with different die types as a secondary resource constraint. Chen and Wu (2006) considered unrelated PMSP with auxiliary equipment constraints as secondary resource constraints. They proposed an effective heuristic based on threshold-accepting methods, tabu lists and improvement procedures to minimize total tardiness. Chaudhry and Drake (2009) developed a search algorithm based on genetic algorithm for identical PMSP with worker assignment. None of these papers addressed resource constrained unrelated parallel machine scheduling problem with machine eligibility restriction. On the other hand most of them considered a partial resource constraint, meaning that, they supposed that there is only one secondary resource which is restricted. Torabi et al. (2013) addressed unrelated parallel machine scheduling problem with non-zero ready times, sequence dependent setup times, and secondary resource constraints. They proposed a multi-objective practical swarm optimization to tackle this problem. However, they supposed that there is only one stock for each secondary resource and this simplification reduces the complexity of resource constrained unrelated parallel machine scheduling problem. A comprehensive survey on parallel machine scheduling problem with additional resources has been presented by Edis et al. (2013).

The remainder of this paper is organized as follows. The next section describes the problem under study and derives a mathematical formulation for considered problem. A new genetic algorithm and a hybrid genetic algorithm to solve the proposed model are clearly elaborated in “Genetic algorithm (GA)” and “Hybrid genetic algorithm (HGA)”, respectively. In “Parameters tuning”, the parameters tuning of the proposed algorithms is presented. In “Computational results”, computational results are presented and finally in “Conclusions and recommendations for future studies” conclusions are provided along with some future research directions.

Problem description and mathematical modeling

In this study, we consider resource-constrained parallel machine scheduling problem in which the machines are unrelated and all of them are available at the beginning of the scheduling and each machine can process only one job at a time. The processing time for each job on each machine is specified. Jobs are available at the beginning of the scheduling and the preemption is not allowed. We assume that each job can only be processed on a specific subset of machines. The setup times are assumed to be part of the processing time. The goal is minimizing the makespan.

Indices

$i = 1, 2, \dots, N$	Index for jobs
$m = 1, 2, \dots, M$	Index for machines
$t = 1, 2, \dots, T$	Index for time periods in the scheduling horizon
$v = 1, 2, \dots, R$	Index for resources

Parameters

P_{im}	Processing time of job i on machine m
Eg_{im}	1 if machine m capable to process job i; 0, otherwise
res_{iv}	Resource requirement of job i to renewable resource v
b_v	Available resource of type v

Decision variables

$x_{imt} = 1$ if job i completes its processing on machine m at time t; otherwise 0

The mathematical model

The RCPMSP with machine eligibility restrictions will be formulated as a 0–1 integer mathematical programming model.

$$Min\ Z = C_{max} \tag{1}$$

Subject to:

$$\sum_{i=1}^{N} \sum_{s=t}^{t+P_{im}-1} x_{ims} \leq 1 \quad \forall m, t \tag{2}$$

$$\sum_{m=1}^{M} \sum_{t=P_{im}}^{T} x_{imt} = 1 \quad \forall i \tag{3}$$

$$\sum_{t=P_{im}}^{T} x_{imt} \leq Eg_{im} \quad \forall m, i \tag{4}$$

$$\sum_{i=1}^{N} \sum_{m=1}^{M} \sum_{s=\mathrm{t}}^{t+P_{im}-1} res_{iv}\ x_{ims} \leq b_v \quad \forall v, t \tag{5}$$

$$\sum_{m=1}^{M} \sum_{t=P_{im}}^{T} x_{imt}.t \leq C_{max} \quad \forall i \tag{6}$$

$$x_{imt} \in \{0, 1\} \quad \forall m, t, i \tag{7}$$

Relation (1) minimizes the makespan. Constraint (2) ensures that on each machine and at each time period, at most one job can be assigned. Constraint (3) imposes that each job should certainly be processed on a machine. Constraint (4) indicates the eligibility restrictions. Constraint (5) imposes that for each resource type v, the total amount assigned to jobs at any time period is less than or equal to the available amount of each resource type b_v. Constraint (6) is used to compute the makespan. Finally, constraint (7) indicates that the decision variables are binary.

Genetic algorithm (GA)

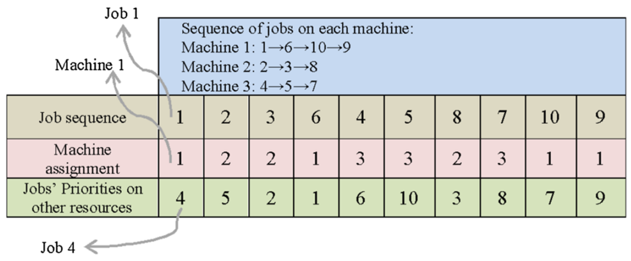
Genetic algorithm (GA) introduced by Holland (1975), is a well-known meta-heuristic approach attempts to mimic the natural evolution process. It has been widely used to solve various optimization problems such as time dependent inventory routing problem (Cho et al. 2014), yard crane scheduling (Liang et al. 2014), optimization for international logistics (Takeyasu and Kainosho 2014), scheduling of JIT cross-docking systems (Fazel Zarandi et al. 2014), straight and U-shaped assembly line balancing (Alavidoost et al. 2014), hybrid flow shop scheduling with single and batch processing machines (Li et al. 2014), etc. In addition many researchers have applied genetic algorithms to solve the parallel machine scheduling problems. Table 1 introduces some of these papers, where the classification scheme of Graham et al. (1979) is used to indicate the problem properties. The comprehensive studies on how to implement the genetic algorithm to solve various single or multi-objective optimization problems have been provided by Gen and Cheng (2000) and Gen et al. (2008).

The GA generally starts with an initial population of individuals (chromosomes), which can either be generated randomly or based on some other algorithms. In each generation, the population goes through the processes of crossover, mutation, fitness evaluation and selection. *Crossover* is the process in which the chromosomes are exchanged in order to create two entirely new chromosomes in a specific pattern. *Mutation* consists of making unexpected changes in the values of one or more genes in a chromosome. A fitness function calculates the fitness of each individual. The *selection* process is that a chromosome with higher fitness value should have a higher chance of selecting into the next

Table 1 The papers which have applied genetic algorithms to solve PMSP

References	Problem
Cheng et al. (1995)	$P_m \,\|\|\, (w_j\, \|c_j - d_D\|)_{Max}$
Sivrikaya-Şerifoglu and Ulusoy (1999)	$P_m \left\| r_i, s_{ij} \right\| W_E \sum_j E_j + W_T \sum_j T_j$
Cochran et al. (2003)	$P_m \,\|\|\, C_{Max}, W_T \sum_j T_j$
Yi and Wang (2003)	$P_m \left\| s_{ij}, batch \right\| W_E \sum_j E_j + W_T \sum_j T_j$
Rajakumar et al. (2006)	$P_m \,\|prec\|\, Workflow\ balancing$
Malve and Uzsoy (2007)	$P_m \,\|r_i, batch, fmls\|\, L_{Max}$
Husseinzadeh Kashan et al. (2008)	$P_m \,\|batch\|\, C_{Max}$
Çakar et al. (2008)	$P_m \,\|prec\|\, \overline{T}$
Damodaran et al. (2009)	$P_m \,\|batch\|\, C_{Max}$
Chaudhry and Drake (2009)	$P_m \,\|worker\ assignment\|\, C_{Max}$
Tavakkoli-Moghaddam et al. (2009)	$R_m \left\| r_i, s_{ijk}, prec \right\| \sum_j U_j, \sum_j C_j$
Vallada and Ruiz (2011)	$R_m \left\| s_{ijk} \right\| C_{Max}$
Li et al. (2011)	$P_m \left\| p_j = \overline{p_j} - x_j, \sum_{j=1}^{n} x_j \le \hat{X} \right\| C_{Max}$
Chang and Cheng (2011)	$R_m \left\| s_{ijk} \right\| C_{Max}$
Alcan and BaşLıGil (2012)	$R_m \,\|fuzzy\ processing\ time\|\, C_{Max}$
Joo and Kim (2012)	$R_m \left\| r_i, s_{ijk} \right\| C_{Max}$
Yilmaz Eroglu et al. (2014)	$R_m \left\| s_{ijk} \right\| C_{Max}$

generation. The new population created in the above manner constitutes the next generation and this process is repeated until a specific stopping criterion is reached. The selection process, mutation and crossover have many models. The population size, the number of generations and the probabilities of mutation and crossover are some other parameters that can be varied to obtain a different genetic algorithm. The overall procedure of the proposed GA is described in following five subsections. First of all an appropriate chromosome representation for considered problem should be illustrated ("Chromosome representation"). Then the GA can be started by generating a set of feasible chromosomes as initial population ("Create initial population") and then all generated chromosomes are evaluated by calculating the objective function value (makespan). Then a pair of chromosomes selected through the parent selection strategy from the current population for crossing that will create a pair of off-springs ("Parent selection strategy"). The proposed crossover mechanism is illustrated clearly in "Design of crossover operator". In crossing procedure $\lfloor Cr * N_{pop}/2 \rfloor$ pair of off-springs should be made and evaluated where Cr is the crossover probability and N_{pop} is the number of population. After that a chromosome is selected completely at random for mutating that will create a mutated off-spring. The proposed mutation mechanism is illustrated in "Design of mutation operator". In mutating



Sequence of jobs on each machine:
Machine 1: 1→6→10→9
Machine 2: 2→3→8
Machine 3: 4→5→7

Job sequence	1	2	3	6	4	5	8	7	10	9
Machine assignment	1	2	2	1	3	3	2	3	1	1
Jobs' Priorities on other resources	4	5	2	1	6	10	3	8	7	9

Fig. 1 Chromosome encoding

procedure $\lfloor Mr * N_{pop} \rfloor$ off-springs should be created and evaluated where Mr is the mutation rate. At the end the current population and the off-springs population are merged and the N_{pop} best of them are transferred to the next generation.

Chromosome representation

The first step in the GA technique is to represent a chromosome to encode the solution of the problem. The chromosome representation has a great influence on the performance of the algorithm. In resource-constrained unrelated parallel machine scheduling problem, a chromosome should simultaneously represent the sequence of jobs on machines and the priority of jobs on other resources. A typical chromosome used in this study is depicted in Fig. 1. The first string is a permutation of the N jobs while the second includes their assigned machine which can process the corresponding job and the third string is the priority of jobs for assigning to the other resources. An example with ($N = 10$, $M = 3$) to illustrate this definition is provided in Fig. 1.

Create initial population

The initial population includes N_{pop} chromosomes. Each chromosome is produced completely at random. However, the third string cannot be produced completely at random. The third string should be generated according to the precedence constraints which are implied by the jobs scheduled to the same machine. The used pseudo code of initialization is indicated in Fig. 2. To generate random solutions for the initial population, the procedure is as follows:

1. To create first string, randomly generate a permutation of N jobs.
2. To create second string, randomly assign an eligible machine to each job of the first string, sequentially.
3. To create third string, first, randomly generate a permutation of N jobs. Then, this string should be repaired according to the precedence constrains which are drawn from the location of jobs in the sequence on each machine.

```
procedure: initialization
$i \leftarrow 0$;
while ($i \le Npop$) do
    for the first string, generate a permutation of $N$ jobs, randomly;
    for the second string, assign an eligible machine to each job;
    for the third string, generate a permutation of $N$ jobs, randomly;
     % Start repair procedure
        create an empty set in the size of $N$ and named it as $pred\_list$;
        for all machine $m$ do
            for all job $j$ on machine $m$ do
                if job $j$ is located on the first position on machine $m$ then
                    $pred\_list\{j\} = \emptyset$;
                else
                    find job $j'$ which is located on previous position of job $j$ ;
                    $pred\_list\{j\} = j'$;
                end if
            end for
        end for
        consider the third string and named it as set $\pi$, consider the $pred\_list$ and generate an empty set in
        the size of $\pi$ and named it as set $\pi'$;
        while ($\pi \neq \emptyset$) do
            for job $j \in \pi$ do
                if $pred\_list\{j\} \in \pi'$ then
                    break ;
                end if
            end for
            add job $j$ to set $\pi'$;
            remove job $j$ from set $\pi$;
        end while
     % End repair procedure                    % the feasible third string $\pi'$ will be obtained
        replace the current third string ($\pi$) with the obtained first string ($\pi'$);
$i \leftarrow i + 1$;
end while
```

Fig. 2 The pseudo code of the initialization procedure

Parent selection strategy

The selection process of GA involves how to choose chromosomes (parents) in the current population for crossing that will create off-springs for the next generation. The *tournament selection* with tour-size = 3 is applied for selecting parents. *Tournament selection* works by selecting a number of chromosomes (tour-size) from the population at random. Then, through the tournament, only the chromosome with the best fitness of those individuals is selected for attending in crossover operation.

Design of crossover operator

The new crossover operator which is employed in this research is similar to *Double-point crossover*. In this operator, firstly choose two chromosomes through *parent selection strategy* from population and call them parents 1 and 2. After that consider the first string of them and randomly choose two crossover points from a discrete uniform distribution in (1, N-1). Then all genes between the crossover points will be copied from parent 1 to the same position of child 2 and similarly from parent 2 to child 1. Parents 1 and 2 elements that will not create any duplication are copied to their exact positions in children 1 and 2, respectively. Then, the remaining empty positions in children 1 and 2 are sequentially filled by the unassigned elements in the order that they appear in the parents 1 and 2, respectively. Similarly, this procedure is applied for the third string of parents. But for each gen of second string of each child, initially generate a random number between [0 1], if it is greater than 0.5 (50 % chance for inheriting from each parent), copy the corresponding machine for that job from parent 1 to child 2 and similarly from the parent 2 to the child 1 otherwise copy the corresponding machine

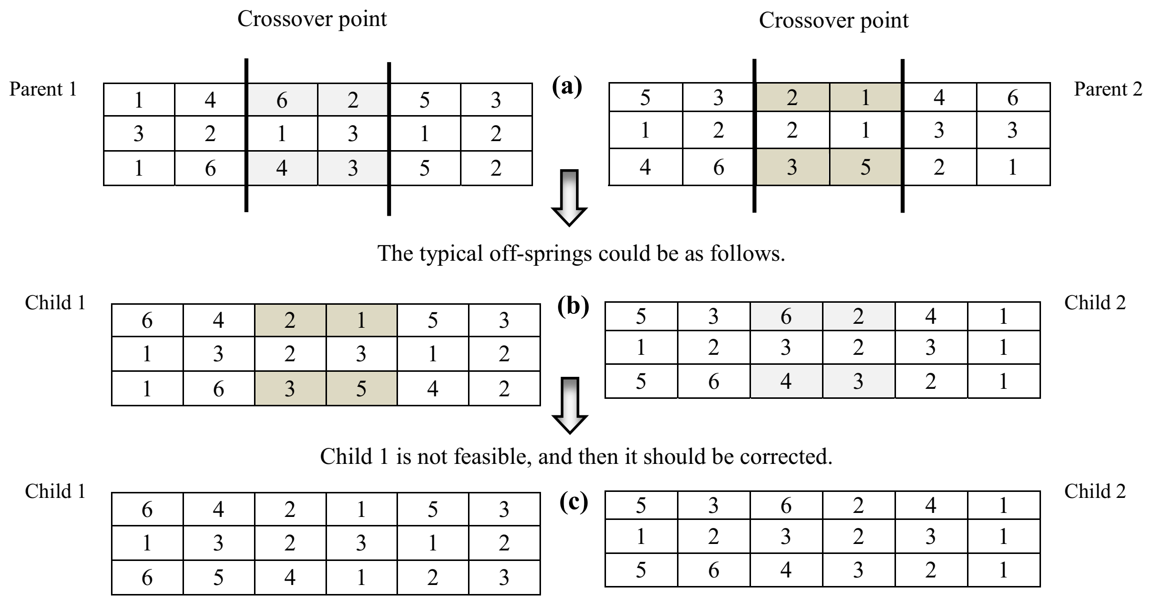
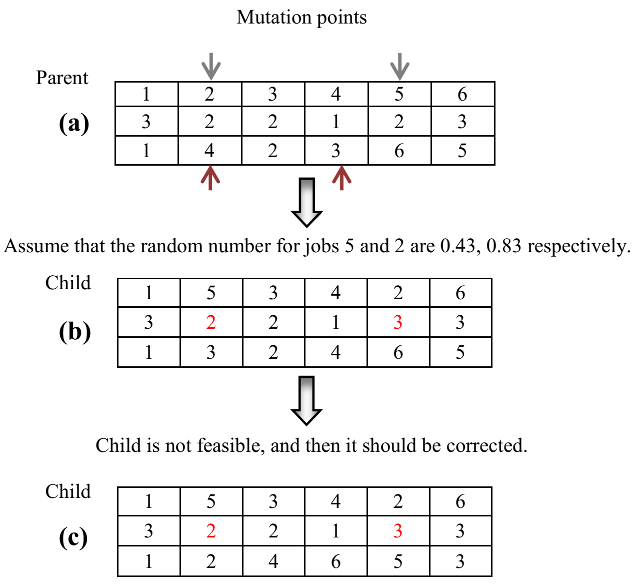


Fig. 3 An illustration of proposed crossover

```
procedure: Crossover
$i \leftarrow 0$;
while $(i \le \lfloor Cr * Npop/2 \rfloor)$ do
    select two chromosomes by selection strategy and call them P1 and P2;
    consider the first and third strings of P1 and P2;
    choose two crossover points;
    copy all genes between crossover points in the same position from P1 to C2 and P2 to C1;
    directly copy the genes of P1 and P2 that will not create any duplication to the same position in C1 and C2, respectively ;
    sequentially fill the remaining empty positions in C1 and C2 by the unassigned elements in the order that they appear in the P1 and P2, respectively;
    for job $j \in$ [the first string of C1] do        % Construct the second string of C1
        $x \leftarrow rand[0\ 1]$;
        if $x > 0.5$ then
            copy the corresponding machine for job $j$ from P2 to C1;
        else
            copy the corresponding machine for job $j$ from P1 to C1;
        end if
    end for
    for job $j \in$ [the first string of C2] do        % Construct the second string of C2
        $x \leftarrow rand[0\ 1]$;
        if $x > 0.5$ then
            copy the corresponding machine for job $j$ from P1 to C2;
        else
            copy the corresponding machine for job $j$ from P2 to C2;
        end if
    end for
        do repair procedure for the third string of C1;
        do repair procedure for the third string of C2;
$i \leftarrow i + 1$;
end while
```

Fig. 4 The pseudo code of the proposed crossover procedure


Mutation points
Parent
(a)
Assume that the random number for jobs 5 and 2 are 0.43, 0.83 respectively.
Child
(b)
Child is not feasible, and then it should be corrected.
Child
(c)

1	2	3	4	5	6
3	2	2	1	2	3
1	4	2	3	6	5

1	5	3	4	2	6
3	2	2	1	3	3
1	3	2	4	6	5

1	5	3	4	2	6
3	2	2	1	3	3
1	2	4	6	5	3

Fig. 5 An illustration of proposed mutation

from parent 1 to child 1 and similarly from parent 2 to child 2. Finally the third string of children 1 and 2 should be repaired based on the obtained first and second strings. Figure 3 demonstrates an example for the crossover operator. For this, consider an example with six jobs and three machines and assume that machine eligibility restrictions are as follows, where M_i is a set of machines which can process job i. Figure 4 indicates the pseudo code of the proposed crossover.

$$M_1 = \{1, 2, 3\} \quad M_2 = \{2, 3\} \quad M_3 = \{1, 2\}$$
$$M_4 = \{1, 2, 3\} \quad M_5 = \{1, 2\} \quad M_6 = \{1, 2, 3\}$$

Design of mutation operator

In this study, an adaptive mutation operator, called *swap* is employed. The proposed mutation mechanism is accomplished through the following steps;

1. Randomly select a chromosome from population and consider first string of it.
2. Randomly choose two gens in the parent.
3. Replace two selected gens.
4. For constructing second string, for all gens (jobs) except replaced gens, copy the corresponding machine from parent to child. But for each replaced gens generate a random number between [0 1], if it is lower than 0.5 (50 % chance for inheriting from his parent) copy the corresponding machine for that job from parent to child, otherwise, ran-

```
procedure: Mutation
 $i \leftarrow 0$;
 while ($i \leq \lfloor Mr * Npop \rfloor$) do
    select a chromosome randomly and call it P;
    consider the first string of P;
    pick up two genes randomly;
    replace two selected gens;
      for job $j \in$ [ [the first string of C]-[the selected gens] ] do   % Construct the second string of Child (C)
          copy the corresponding machine for job $j$ from P to C;
      end for
      for job $j \in$ [the selected gens] do
          $x \leftarrow rand[0\ 1]$;
          if $x > 0.5$ then
             randomly select a machine from eligible machine set where the selected machine should
             differ from previous machine;
          else
              copy the corresponding machine for job $j$ from P to C;
          end if
      end for
    consider the third string of P;
    pick up two genes randomly;
    replace two selected gens;
        do repair procedure for the third string of C;
 $i \leftarrow i + 1$;
end while
```

Fig. 6 The pseudo code of the proposed mutation procedure

domly select a machine from the eligible machines set where the selected machine should differ from previous one.

5. Do the steps 1–3 for the third string and then it should be repaired based on the obtained first and second strings.

Figure 5 illustrates an instance for the proposed mutation operator with regard to the pervious example. Moreover, the used pseudo code for the proposed mutation is indicated in Fig. 6.

Hybrid genetic algorithm (HGA)

As stated before, the problem considered in this study is the combination of two optimization problems including scheduling and resource allocation. In GA to tackle this combinatorial optimization problem a three level solution encoding scheme is proposed. In order to find how effective our proposed GA and their operators work and besides, because no benchmark problems are available in the literature to confirm and compare the results obtained by GA, a new genetic algorithm HGA is suggested to solve the experimental problems. In HGA, instead of employing a three level chromosome, we developed a genetic algorithm along with a heuristic procedure which performs the search procedure by a two level solution encoding scheme (ignoring the third string). For calculating the objective function value of each chromosome in HGA, at first according to the acquired sequence of jobs on machines the start and completion times of jobs are computed by considering the machine as only resource which is restricted. In the second step, to obtain a feasible order of jobs for allocating to the other resources, jobs are sorted based on their start times. In using this procedure, for the jobs which have the same start times, the job with longer processing time is scheduled first. After assigning the jobs to the other resources based on acquired order, the final objective function value can be calculated. The overall procedure for HGA is the same as the overall procedure for GA and the mutation and the crossover operators which are used for GA are adopted for HGA. Now, we give an instance with six jobs, two machines and two additional resources to help

Table 2 Sample problem data

Number of jobs = 6; number of machines = 2; number of resources = 2							
Job	1	2	3	4	5	6	
P_{i1}	11	–	10	6	14	8	
P_{i2}	8	13	16	2	–	16	
R_{i1}	1	2	1	1	2	1	$b_1 = 2$
R_{i2}	1	1	1	0	1	0	$b_2 = 1$

Table 3 A typical chromosome for HGA

1	2	4	6	5	3
1	2	1	2	1	2

Table 4 The start and finish times of jobs in step 1

Jobs	1	2	3	4	5	6
ST_i	0	0	29	11	17	13
FT_i	11	13	**45**	17	31	29

The bold value shows the maximum completion time

Table 5 The start and finish times of jobs in last step

Jobs	1	2	3	4	5	6
ST_i	13	0	44	24	30	13
FT_i	24	13	**60**	30	44	29

The bold value shows the maximum completion time

Table 6 Parameters and their ranges

Parameter	Level
It	(100–300)
N_{pop}	(40–60)
C_r	(0.6–0.9)
M_r	(0.1–0.4)

in illustrating how our proposed HGA works. The processing times and the resource requirements of jobs are shown in Table 2. The machine eligibility constraints are as follows:

$$M_1 = \{1, 2\} \quad M_2 = \{2\} \quad M_3 = \{1, 2\}$$
$$M_4 = \{1, 2\} \quad M_5 = \{1\} \quad M_6 = \{1, 2\}$$

In order to find the objective function value of the chromosome depicted in Table 3, at first the start and completion times of each job are evaluated without considering the resource constraints. Table 4 shows the start time (ST) and the finish time (FT) of each job after the first step with $C_{max} = 45$.

In step 2, jobs are sorted based on their start times. There is a tie between job1 (J1) and job2 (J2), and tie is broken in favor of a job with longer processing time. As a result the feasible order of jobs for allocating to the other resources is J2–J1–J4–J6–J5–J3. With this sequence, jobs will be assigned to their resources. Finally, the start and finish times of jobs are recalculated by considering both machines and additional resources. The first job in the sequence is J2 on machine 2 (M2). This job needs two units of resource 1 (R1) and one unit of resource 2 (R2). Both resources are available at time zero so J2 can be started at time zero and the finish time of J2 is 0 + 13 = 13. The second job in the sequence is J1 on M1. J1 needs one unit of R1 and one unit of R2. Both R1 and

R2 are unavailable until at time 13. So the earliest start time for J1 is 13 and the finish time of J1 is 13 + 11 = 24. This procedure is continued until all jobs are scheduled. The final start and finish times of jobs are shown in Table 5. As can be seen in Table 5 the objective function value is increased considerably from 45 to 60.

Parameters tuning

In this section, design of experiment (DOE) approach is employed to investigate the effect of different levels of factors on the performance of the proposed algorithms. Hence, in order to calibrate the parameters, Response Surface Methodology (RSM) is applied in DOE approach. RSM is a technique for determining and representing the cause-and-effect relationship between true mean responses and input control variables influencing the responses as a two-or three-dimensional hyper surface (Montgomery 1991). The GA factors are: number of iterations (It), the number of population (N_{pop}), the crossover rate (C_r) and the mutation rate (M_r). The search ranges and the different levels of the parameters are shown in Table 6.

In proposed GA algorithms because four parameters exist, we use a fractional factorial central composite face-centered design with 2^{4-1} factorial points, 2×4 axial points and 4 center points, totally 20 experiments. For each experiment five problems with various sizes are implemented. The results are normalized by relative percentage deviation (RPD) method and the average value of RPDs is reported for each test as a final response and a quadratic model will be fitted on the acquired responses. The obtained regression model is optimized within the range of the parameters by using LINGO 9.0 software and the optimum combinations of the parameters are shown in Table 7 for each algorithm.

Computational results

To solve the presented model and evaluate the performance of the proposed GAs, computational experiments are conducted using randomly generated test problems. To generate random processing times, number of resources and the available amount of each resource, we use integer uniform distribution (IUD) [1,20], [1,5] and [1,5], respectively. To produce the eligible constraints, initially generate an (N, M)-dimensional binary matrix, namely (Eg), in which in any row of this matrix, at least, there are $\left[\frac{M}{2}\right]$ elements 1. When $Eg\,(i, m) = 1$, it means that machine m capable to process job i. To validate the proposed mathematical model and evaluate the optimality of the proposed meta-heuristics, a numerical example with eight jobs, three machines and a single additional resource is provided as a benchmark problem. The input parameters related to this example is given in Table 8. This example is solved optimally by LINGO 9.0 software and the proposed GAs with makespan 55. The Gantt chart for the optimal solution is provided in Fig. 7.

Table 7 Exact values of parameters

Method	Parameters	Optimum value
GA	It	250
	N_{pop}	50
	C_r	0.75
	M_r	0.25
HGA	It	300
	N_{pop}	60
	C_r	0.75
	M_r	0.25

Table 8 Input parameters of example

Job	1	2	3	4	5	6	7	8	
P_{i1}	10	12	8	4	–	12	11	–	
P_{i2}	15	8	10	6	7	14	–	12	
P_{i3}	8	10	12	–	12	17	9	14	
R_{i1}	2	1	2	1	1	2	2	1	$b_1 = 2$

The proposed model is applied for 25 test problems including 10 small and 15 large problems. We solve the small problems with two approaches: the optimal solution approach under LINGO 9.0 software and the proposed genetic algorithms. However, it is impossible to obtain an optimal solution for all of the problems in a reasonable CPU time, therefore the problems are optimally solved by the branch-and-bound approach (B&B) under the LINGO 9.0 software with 3 h run time limitation. Thus, the best solution obtained after 3 h is reported for the small size problems that cannot be optimally solved by the B&B in a reasonable CPU time. All examples are also solved by the proposed GAs and each algorithm is run five times for each problem and the obtained results compared with each other in terms of the average relative percentage deviation (ARPD) and CPU times. RPD is computed by the following formula;

$$RPD = \frac{C_{max}^{alg} - C_{max}^{min}}{C_{max}^{min}} \times 100 \tag{8}$$

where C_{max}^{alg} is the makespan obtained by each algorithm for a given problem and C_{max}^{min} is the best value of total makespans obtained by through algorithms for the related problem. The proposed algorithms were coded in *MATLAB R2012a* and run on a personal computer including Intel(R) Core(TM) i5 CPU with 2.27 GHz speed and 3 GB of RAM.

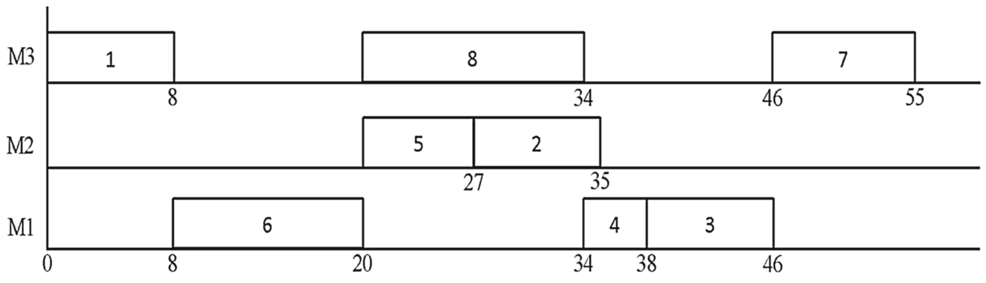


Fig. 7 The Gantt chart of optimal schedule

Table 9 Comparison between the exact approach and GAs for small-size problems

No.	Problem information	ARPD			CPU time(s)		
	N × M	Lingo	GA	HGA	Lingo	GA	HGA
1	4 × 2	0	0	0	1	6	4
2	6 × 2	0	0	0	5	7	5
3	6 × 3	0	0	0	8	6	5
4	8 × 2	0	0	0	18	7	5
5	8 × 3	0	0	0	267	9	6
6	8 × 4	0	0	0	1535	12	8
7	10 × 2	0	0	0	5537	8	6
8	10 × 3	0	0	0	8632	10	8
9	10 × 4	0	0	0	10,800	16	10
10	12 × 4	11.5	**0**	**0**	10,800	20	12

The value of superior algorithm are shown in bold value

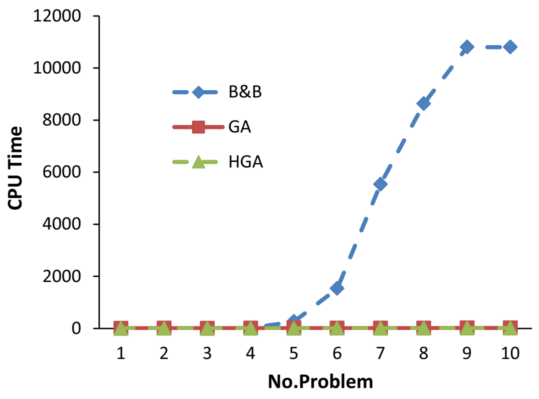


Fig. 8 CPU times reported by B&B method and GAs for small problems

Table 10 ARPD for the problems in large scale

No.	$N \times M$	ARPD	
		GA	HGA
1	20 × 2	0	0
2	20 × 4	0	0
3	20 × 6	0	0
4	30 × 2	0	0
5	30 × 4	0	0
6	30 × 6	**0. 44**	0. 59
7	40 × 4	0	0
8	40 × 6	0	0
9	40 × 8	**0. 97**	2.06
10	50 × 6	**0**	0.38
11	50 × 8	**0. 59**	0.84
12	50 × 10	0. 53	**0.26**
13	60 × 6	**0.19**	0.45
14	60 × 8	**0.18**	0.36
15	60 × 10	**0**	1.39

The value of superior algorithm are shown in bold value

Table 9 shows the obtained results from B&B, GA and HGA for 10 small size problems. As it can be seen from Table 9, for these problems the proposed GAs are able to get exact solutions for all of problems in reasonable CPU times. However, the LINGO 9.0 software could not achieve best solution as well as GAs for problem 10 in 3 h run time limitation. Figure 8 shows a polynomial behavior of CPU times by the proposed GAs against an exponential behavior of B&B method by the increase of the problem size for small problems. As you can see in Fig. 8, the GAs spend less CPU times than B&B for achieving to an optimal solution.

In order to create more challenging comparison among GAs, we generated 15 large test problems and solved them five times by each proposed algorithm. The computational results compared with each other in terms of ARPD. The average values of RPDs (ARPD) for large problems are

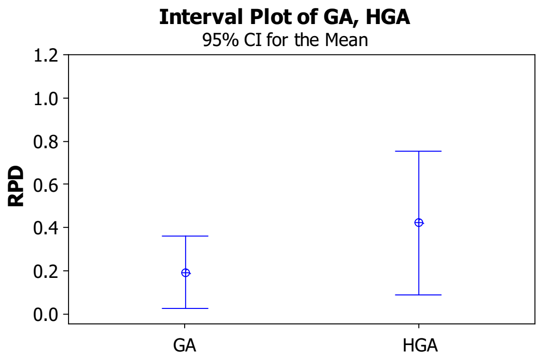


Fig. 9 Means plot and LSD intervals for algorithms in large test problems

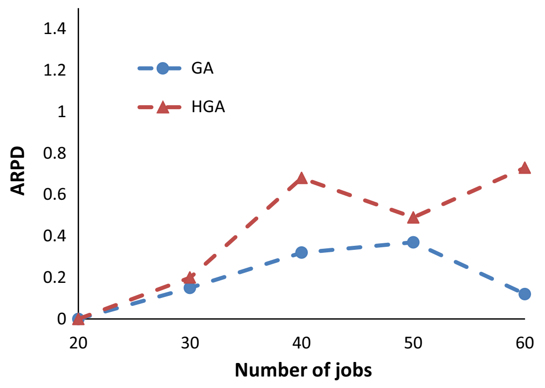


Fig. 10 Interaction plot between the performance of algorithms and number of jobs in terms of ARPD

reported in Table 10. In order to create significant statistical analysis of the comparison among algorithms, the means plot and least significant difference (LSD) intervals (at the 95 % confidence level) for ARPD values of the algorithms for large size problems is shown in Fig. 9.

According to Naderi and Ruiz (2010) which noticed that overlapping confidence intervals between any two means indicate that there is no statistically significant difference between them. With regard to this fact and as can be seen in Fig. 9, there is no statistically significant difference between GA and HGA. However, as you can see from Fig. 9, GA has more central tendency and lower variability of ARPD. As a result, for selecting only one algorithm, GA is more reliable than HGA to finding more accurate makespan. It is worth noting that GA spends approximately double run times against HGA for large scale problems.

In this section, some sensitive analyses are performed for assessing the behaviour of algorithms versus different conditions. Figures 10 and 11 indicate the interaction plot between the algorithms performance and number of jobs, number of machines in terms of ARDP, respectively. As it can be observed in Figs. 10 and 11, the GA has better performance in all conditions for large size problems; however, this transcendence is not tangible.

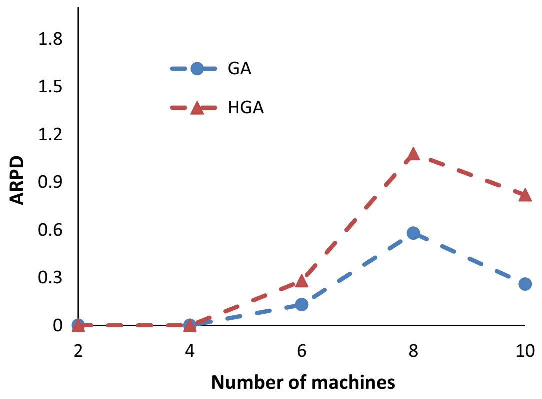


Fig. 11 Interaction plot between the performances of algorithms and number of machines in terms of ARPD

Conclusions and recommendations for future studies

In this study we considered an unrelated parallel machine scheduling problem with machine eligibility and resource constraints. To solve this problem, two different approaches were proposed. The first approach was based on an integer programming model; however the mathematical model is not tractable for large size problems. As a result, in order to find proper schedules that minimize makespan, for second approach, we studied two structures of genetic algorithm, a pure genetic algorithm (GA) and a genetic algorithm along with a heuristic procedure (HGA). To assess the effectiveness of the proposed algorithms some random test problems in two scales were generated and the obtained results were analyzed and compared with each other. The comparisons have revealed that the proposed GA had better performance against HGA for solving the considered test problems, but this priority was not noticeable. On the other hand, the results demonstrated that our proposed heuristic algorithm is effective in both small and large size problems. For the sake of completeness, it should be noted that the proposed HGA could make desirable solutions, not as well as GA, for the considered test problems by spending less CPU times. As an interesting future research, it is worthwhile to develop the problem by adding some practical assumptions including machine breakdown, parallel batch processing and sequence dependent setup time. Furthermore, it is recommended to apply other efficient heuristics or meta-heuristic algorithms for solving the problem.

Appendix

The data set and the computational results of the large problem 15 which contains 60 jobs and 10 machines are as follows:

See Tables 11 and 12, Fig. 12.

Table 11 The data set of problem 15

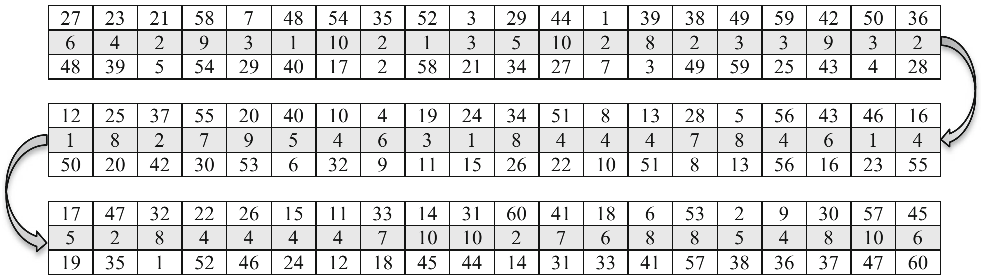
Jobs	Processing times										Resources	
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	R1	R2
1	4	2	14	6	3	9	11	8	10	12	2	1
2	–	–	9	8	6	8	13	7	8	8	3	1
3	20	16	8	12	14	17	9	17	10	5	3	1
4	19	10	17	–	18	7	10	–	12	13	2	3
5	5	16	–	18	7	–	–	5	11	10	3	2
6	20	9	19	11	7	13	7	5	17	11	3	2
7	13	17	4	8	7	13	10	–	13	11	3	2
8	–	7	20	3	4	15	6	20	9	4	2	3
9	13	–	15	9	12	15	15	11	18	12	2	3
10	10	18	–	5	16	–	12	9	6	12	3	3
11	18	12	5	2	13	10	6	12	14	20	3	3
12	5	–	6	12	8	–	14	16	10	12	2	3
13	16	–	17	10	–	14	12	13	11	18	3	3
14	–	17	13	16	12	17	11	–	13	10	3	3
15	8	12	–	7	13	12	12	9	14	8	1	2
16	6	4	11	3	4	8	–	4	8	9	1	2
17	14	20	16	20	9	17	15	–	11	–	2	3
18	–	9	12	8	11	6	10	4	17	19	1	2
19	15	12	6	9	18	10	10	15	10	16	2	2
20	9	9	19	15	–	6	19	11	5	6	1	3
21	17	9	–	–	15	17	11	–	13	16	2	1
22	17	12	17	9	11	–	18	14	–	10	3	2
23	6	12	11	6	12	6	13	19	9	18	1	2
24	5	10	11	6	9	20	17	14	15	15	2	2
25	11	5	7	16	17	6	–	2	19	–	3	2
26	11	9	12	6	18	8	20	17	11	13	2	3
27	16	14	10	17	16	3	15	11	4	12	3	3
28	18	18	17	5	9	11	6	14	8	11	2	1
29	9	12	11	6	3	7	11	10	5	–	3	1
30	13	18	13	17	17	9	–	6	–	11	3	3
31	–	8	–	12	11	4	19	8	18	3	2	2
32	18	5	17	–	20	10	–	5	14	–	2	1
33	15	–	–	20	11	18	6	15	15	17	2	2
34	10	14	5	8	12	9	–	3	–	17	1	3
35	6	2	18	17	18	15	13	7	18	16	1	1
36	–	5	14	–	–	20	9	18	14	8	3	2
37	9	4	12	20	–	9	4	4	13	20	2	3
38	9	2	10	7	16	–	7	19	12	12	1	1
39	15	4	6	4	5	18	20	2	–	18	2	2
40	15	19	20	9	3	11	6	6	10	19	3	3
41	19	11	10	10	9	3	2	16	9	9	2	3
42	–	19	10	11	13	14	–	9	1	12	3	3
43	3	19	–	5	16	2	20	4	5	20	2	3
44	–	7	3	8	1	2	6	8	1	2	1	2

Table 11 continued

Jobs	Processing times										Resources	
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	R1	R2
45	–	13	15	17	11	5	12	15	15	–	2	1
46	5	19	10	6	14	19	19	10	16	10	2	3
47	9	1	7	2	4	9	7	2	–	6	2	1
48	2	8	11	–	7	13	6	8	4	–	2	2
49	–	–	7	10	14	16	9	13	14	10	2	1
50	5	4	3	13	14	9	–	12	13	18	3	1
51	–	11	15	7	10	17	7	17	8	10	3	3
52	2	18	14	10	5	17	4	18	10	8	1	1
53	11	4	6	7	–	9	11	2	7	19	3	3
54	11	15	–	12	12	10	14	–	17	3	2	3
55	8	11	8	12	–	8	3	16	12	20	3	2
56	6	12	10	5	7	–	6	18	8	9	1	2
57	19	14	9	13	10	17	12	15	–	6	1	1
58	13	7	14	13	10	3	13	5	2	12	2	1
59	10	8	7	14	10	17	9	9	20	9	3	2
60	–	1	7	7	5	2	–	1	9	2	3	2
					Available number of each resource						3	3

Table 12 The computational result of the best solution obtained by GA with C_{max} = 234

Job	Start time	Finish time	Job	Start time	Finish time	Job	Start time	Finish time	Job	Start time	Finish time
1	141	143	16	39	42	31	212	215	46	182	187
2	27	33	17	18	27	32	136	141	47	232	233
3	54	62	18	87	93	33	215	221	48	0	2
4	80	87	19	176	182	34	44	47	49	62	69
5	4	9	20	168	173	35	67	69	50	173	176
6	154	159	21	35	44	36	223	228	51	109	116
7	50	54	22	121	130	37	228	232	52	2	4
8	106	109	23	33	39	38	176	178	53	159	161
9	145	154	24	187	192	39	2	4	54	9	12
10	116	121	25	76	78	40	15	18	55	93	96
11	143	145	26	130	136	41	221	223	56	62	67
12	192	197	27	47	50	42	167	168	57	215	221
13	96	106	28	87	93	43	78	80	58	33	35
14	202	212	29	12	15	44	42	44	59	69	76
15	136	143	30	161	167	45	197	202	60	233	**234**



27	23	21	58	7	48	54	35	52	3	29	44	1	39	38	49	59	42	50	36
6	4	2	9	3	1	10	2	1	3	5	10	2	8	2	3	3	9	3	2
48	39	5	54	29	40	17	2	58	21	34	27	7	3	49	59	25	43	4	28

12	25	37	55	20	40	10	4	19	24	34	51	8	13	28	5	56	43	46	16
1	8	2	7	9	5	4	6	3	1	8	4	4	4	7	8	4	6	1	4
50	20	42	30	53	6	32	9	11	15	26	22	10	51	8	13	56	16	23	55

17	47	32	22	26	15	11	33	14	31	60	41	18	6	53	2	9	30	57	45
5	2	8	4	4	4	4	7	10	10	2	7	6	8	8	5	4	8	10	6
19	35	1	52	46	24	12	18	45	44	14	31	33	41	57	38	36	37	47	60

Fig. 12 The chromosome of the best solution obtained by GA

References

Alavidoost, M. H., Fazel Zarandi, M. H., Tarimoradi, M., & Nemati, Y. (2014). Modified genetic algorithm for simple straight and U-shaped assembly line balancing with fuzzy processing times. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-014-0978-4

Alcan, P., & BaşLıGil, H. S. (2012). A genetic algorithm application using fuzzy processing times in non-identical parallel machine scheduling problem. *Advances in Engineering Software*, *45*(1), 272–280.

Arnaout, J.-P., Musa, R., & Rabadi, G. (2014). A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines—part II: Enhancements and experimentations. *Journal of Intelligent Manufacturing*, *25*(1), 43–53.

Bilyk, A., & Monch, L. (2012). A variable neighborhood search approach for planning and scheduling of jobs on unrelated parallel machines. *Journal of Intelligent Manufacturing*, *23*(5), 1621–1635.

Çakar, T., Köker, R., & Demir, Hİ. (2008). Parallel robot scheduling to minimize mean tardiness with precedence constraints using a genetic algorithm. *Advances in Engineering Software*, *39*(1), 47–54.

Centeno, G., & Armacost, R. L. (1997). Parallel machine scheduling with release time and machine eligibility restrictions. *Computers and Industrial Engineering*, *33*(1–2), 273–276.

Centeno, G., & Armacost, R. L. (2004). Minimizing makespan on parallel machines with release time and machine eligibility restrictions. *International Journal of Production Research*, *42*(6), 1243–1256.

Chaudhry, I. A., & Drake, P. R. (2009). Minimizing total tardiness for the machine scheduling and worker assignment problems in identical parallel machines using genetic algorithms. *International Journal of Advanced Manufacturing Technology*, *42*(5–6), 581–594.

Chen, J. F. (2005). Unrelated parallel machine scheduling with secondary resource constraints. *International Journal of Advanced Manufacturing Technology*, *26*(3), 285–292.

Chen, J. F., & Wu, T. H. (2006). Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. *Omega*, *34*(1), 81–89.

Cheng, R., Gen, M., & Tozawa, T. (1995). Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithm. *Computers and Industrial Engineering*, *29*(1–4), 513–517.

Cheng, R., & Gen, M. (1997). Parallel machine scheduling problems using memetic algorithms. *Computers and Industrial Engineering*, *33*(3–4), 761–764.

Cho, D. W., Lee, Y. H., Lee, T. Y., & Gen, M. (2014). An adaptive genetic algorithm for the time dependent inventory routing problem. *Journal of Intelligent Manufacturing*, *25*(5), 1025–1042.

Cochran, J. K., Horng, S. M., & Fowler, J. W. (2003). A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers and Operations Research*, *30*(7), 1087–1102.

Damodaran, P., Hirani, N. S., & 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.

Edis, E. B., & Ozkarahan, I. (2012). Solution approaches for a real-life resource-constrained parallel machine scheduling problem. *International Journal of Advanced Manufacturing Technology*, *58*(9–12), 1141–1153.

Edis, E. B., Oguz, C., & Ozkarahan, I. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, *230*, 449–463.

Fazel Zarandi, M.H., Khorshidian, H., & Akbarpour Shirazi, M. (2014). A constraint programming model for the scheduling of JIT cross-docking systems with preemption. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-013-0860-9

Gen, M., & Cheng, R. (2000). *Genetic algorithms and engineering optimization*. New York: Wiley.

Gen, M., Cheng, R., & Lin, L. (2008). *Network models and optimization: Multiobjective genetic algorithm approach*. London: Springer.

Chang, P. C., & Cheng, S. H. (2011). Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. *Applied Soft Computing*, *11*(1), 1263–1274.

Gokhale, R., & Mathirajan, M. (2012). Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. *International Journal of Advanced Manufacturing Technology*, *60*(9–12), 1099–1110.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, *5*, 287–326.

Holland, H. J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, Michigan: The University of Michigan Press.

Hu, X., Bao, J. S., & Jin, Y. (2010). Minimising makespan on parallel machines with precedence constraints and machine eligibility restrictions. *International Journal of Production Research*, *48*(6), 1639–1651.

Husseinzadeh Kashan, A., Karimi, B., & Jenabi, M. (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers and Operations Research*, *35*(4), 1084–1098.

Joo, C. M., & Kim, B. S. (2012). Parallel machine scheduling problem with ready times, due times and sequence-dependent setup times using meta-heuristic algorithms. *Engineering Optimization*, *44*(9), 1021–1034.

Kim, D. W., Kim, K. H., Jang, W., & Frank Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer Integrated Manufacturing*, *18*(3–4), 223–231.

Li, D., Meng, X., Liang, Q., & Zhao, J. (2014). A heuristic-search genetic algorithm for multi-stage hybrid flow shop scheduling with single processing machines and batch processing machines. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-014-0874-y

Li, K., Shi, Y., Yang, S. L., & Cheng, B. Y. (2011). Parallel machine scheduling problem to minimize the makespan with resource dependent processing times. *Applied Soft Computing*, *11*(8), 5551–5557.

Liang, C. J., Chen, M., Gen, M., & Jo, J. (2014). A multi-objective genetic algorithm for yard crane scheduling problem with multiple work lines. *Journal of Intelligent Manufacturing*, *25*(5), 1013–1024.

Liao, L. W., & Sheen, G. J. (2008). Parallel machine scheduling with machine availability and eligibility constraints. *European Journal of Operational Research*, *184*(2), 458–467.

Malve, S., & Uzsoy, R. (2007). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers and Operations Research*, *34*(10), 3016–3028.

Mehravaran, Y., & Logendran, R. (2013). Non-permutation flowshop scheduling with dual resources. *Expert Systems with Applications*, *40*(13), 5061–5076.

Montgomery, D. C. (1991). *Design and analysis of experiments* (3rd ed.). New York: Wiley.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flow shop scheduling problem. *Computers and Operations Research*, *37*(4), 754–768.

Pan, Q. K., Wang, L., Mao, K., Zhao, J. H., & Zhang, M. (2013). An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. *IEEE Transactions on Automation Science and Engineering*, *10*(2), 307–322.

Pinedo, M. (1995). *Scheduling theory, algorithms and systems*. Upper Saddle River: Prentice Hall.

Rajakumar, S., Arunachalam, V. P., & Selladurai, V. (2006). Workflow balancing in parallel machine scheduling with precedence constraints using genetic algorithm. *Journal of Manufacturing Technology Management*, *17*(2), 239–254.

Sheen, G. J., Liao, L. W., & Lin, C. F. (2008). Optimal parallel machines scheduling with machine availability and eligibility constraints. *International Journal of Advanced Manufacturing Technology*, *36*(1–2), 132–139.

Silva, C., & Magalhaes, J. M. (2006). Heuristic lot size scheduling on unrelated parallel machines with applications in the textile industry. *Computer and Industrial Engineering*, *50*(1–2), 76–89.

Sivrikaya-Şerifoglu, F., & Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research*, *26*(8), 773–787.

Takeyasu, K., & Kainosho, M. (2014). Optimization technique by genetic algorithms for international logistics. *Journal of Intelligent Manufacturing*, *25*(5), 1043–1049.

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

Torabi, S. A., Sahebjamnia, N., Mansouri, S. A., & Aramon Bajestani, M. (2013). A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem. *Applied Soft Computing*, *13*(12), 4750–4762.

Vallada, E., & 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.

Ventura, J. A., & Kim, D. (2000). Parallel machine scheduling about an unrestricted due date and additional resource constraints. *IIE Transactions*, *32*(2), 147–153.

Ventura, J. A., & Kim, D. (2003). Parallel machine scheduling with earliness-tardiness penalties and additional resource constraints. *Computers and Operations Research*, *30*(13), 1945–1958.

Wang, I. L., Wang, Y. C., & Chen, C. W. (2013). Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics. *Flexible Services and Manufacturing Journal*, *25*(35), 343–366.

Yi, Y., & Wang, D. W. (2003). Soft computing for scheduling for with batch setup times and earliness-tardiness penalties on parallel machines. *Journal of Intelligent Manufacturing*, *14*, 311–322.

Yilmaz Eroglu, D., Ozmutlu, H. C., & Ozmutlu, S. (2014). Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *International Journal of Production Research*, *52*(19), 5841–5856.

Yu, L., Shih, H. M., Pfund, M., Carlyle, W. M., & Fowler, J. W. (2002). Scheduling of unrelated parallel machines: An application to PWB manufacturing. *IIE Transactions*, *34*(11), 921–931.