Indonesian Journal of Electrical Engineering and Computer Science
Vol. 26, No. 2, May 2022, pp. 1036~1049
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v26.i2.pp1036-1049

Guided genetic algorithm for solving unrelated parallel machine scheduling problem with additional resources

Munther H. Abed, Mohd Nizam Mohmad Kahar
Faculty of Computing, College of Computing and Applied Sciences, University Malaysia Pahang, Pahang, Malaysia

Article Info

Article history:

Received Sep 21, 2021
Revised Mar 5, 2022
Accepted Mar 15, 2022

Keywords:

Genetic algorithm
Makespan
Metaheuristics
Resource constraints
Unrelated parallel machine scheduling problem

ABSTRACT

This paper solved the unrelated parallel machine scheduling with additional resources (UPMR) problem. The processing time and the number of required resources for each job rely on the machine that does the processing. Each job j needed units of resources (rjm) during its time of processing on a machine m. These additional resources are limited, and this made the UPMR a difficult problem to solve. In this study, the maximum completion time of jobs makespan must be minimized. Here, we proposed genetic algorithm (GA) to solve the UPMR problem because of the robustness and the success of GA in solving many optimization problems. An enhancement of GA was also proposed in this work. Generally, the experiment involves tuning the parameters of GA. Additionally, an appropriate selection of GA operators was also experimented. The guide genetic algorithm (GGA) is not used to solve the unspecified dynamic UPMR. Besides, the utilization of parameters tuning and operators gave a balance between exploration and exploitation and thus help the search escape the local optimum. Results show that the GGA outperforms the simple genetic algorithm (SGA), but it still didn't match the results in the literature. On the other hand, GGA significantly outperforms all methods in terms of CPU time.

Corresponding Author:

Munther H. Abed
Faculty of Computing, College of Computing and Applied Sciences, University Malaysia Pahang
Pahang, Malaysia
Email: munt1979@yahoo.com

1. INTRODUCTION

Technological advancements have influenced the integration of artificial intelligence (AI) into manufacturing industries operations such as robotics, production scheduling. These integrations help to minimize the manufacturing production operational cost and increase the revenue without reducing the product quality for customer satisfaction. The production scheduling refers to a process of assigning jobs to machines with the aim to optimize some performance measures. An example of production scheduling is the unrelated parallel machine scheduling problem (UPM). UPM aims to schedule all jobs with a minimum completion time (also known as makespan, C_{max}).

The UPM problem has several variants with differences in its constraints [1], [2]. An example includes an unrelated parallel machine with sequence dependent setup time (UPMSP) [3]-[5] and an unrelated parallel machine scheduling problem with additional resources (UPMR) [1], [6]-[8]. In UPMSP, each machine has its own specific matrix of setup times for all jobs. While, in UPMR, the additional resources (i.e., constraints) must be taken into account when scheduling jobs to machine. The additional resources in UPMR refer to materials, human labor, tools, fixtures, and industrial robots. UPMR problem can

be considered as one of the most important combinatorial optimization problems as it closely resembles the real-world problem. This research work focuses on solving UPMR problems.

Various researchers have reported solving the UPMR problem using different approaches. Some of them had used exact method, heuristic and approximate approaches [1], [9]-[11]. The exact methods are suitable for small size problems, since they seek all possible solutions in the entire search space [12]. However, for large size problems, an exact method is impractical due to requiring a tremendous amount of computational time. Hence, an exact method is not suitable to solve such optimization problems [13], [14]. Over the years, metaheuristic algorithms have shown the ability to solve wide optimization problems [15]-[19]. According to [20], genetic algorithm (GA) is able to solve several continuous optimization problems, such as exam timetabling problem [21], face recognition system [22], [23], travelling salesman problem [24], medical robot [25] and many others. GA ability to solve a variety of problems and to produce good quality solution attracts us to explore it into solving UPMR problems. In [26], they implemented GA into solving static and specified UMPR problems. However, in this work, dynamic and unspecified problems in UPMR are considered. Many works on UPMR have been reported in the literature to minimize the makespan (see Table 1).

Table 1. A summary of methods applied for UPMR problem

Method	Reference	Description
A heuristic approach	[27]	Specified dynamic
Two-phased LP rounding technique $(4 + 2\sqrt{2})$ and $(3 + 2\sqrt{2})$	[28]	Unspecified static
Deterministic 3/2-approximation, 2-approximation and 4-approximation method	[29]	Unspecified dynamic
A heuristic approach	[30]	Specified dynamic
A 3.75-approximation algorithm	[30]	Unspecified dynamic
A Lagrangian-based CP method	[31]	Unspecified dynamic
CP model, IP model, and integrated IP/CP	[32]	Specified dynamic
IP model, a relaxed IP based CP model and IP/CP model	[33]	Unspecified static & dynamic
IP/IP model and IP/CP model	[34]	Specified dynamic
An ILP program and a two-phase approach	[11]	Unspecified dynamic
Matheuristic strategies	[1]	Unspecified dynamic
Genetic algorithm and hybrid genetic algorithm	[26]	Specified static
MILP model and CP model	[35]	Unspecified dynamic
Local search methods and multi-pass heuristics	[8]	Unspecified dynamic
CP model	[36]	Unspecified dynamic
Enriched scatter search and enriched iterated greedy	[37]	Unspecified dynamic

As seen in the literature, GA shows the ability to solve numerous problems and produce quality solution and it has many novel characteristic [38]-[40]. Hence, it is clear that GA is a robust algorithm. However, it appears that there is a lack of work in GA that is projected into solving the UPMR problem. In this work, we will investigate GA into solving the unspecified dynamic UPMR problem. Detailed description of the UPMR problem follows in the next section. The rest of the paper is organized as shown in: Section 2 describes the UMPR problem in detail. The standard GA algorithm (referred to as SGA) and the enhancement of GA (referred to as GGA) are discussed in section 3. The experimental result and the discussion are described in section 4. Finally, the conclusion and future work are discussed in section 5.

2. PROBLEM DESCRIPTION

UPMR problem requires scheduling a set of jobs *J* without interruption (preemption is forbidden), into unrelated machines, *M* [35]. Indeed, the required number of resources need to be fulfilled. These resources are limited and it is important to take into consideration when assigning jobs for processing at the machines. The allocation of resources to machines can be divided into two types [36] that includes static [9], [26] and dynamic [1], [8], [28], [31], [37]. In static, the allocation of resources to machines is fixed during the whole-time horizon and vice versa in the dynamic. The additional resources are divided into: renewable, non-renewable and doubly constrained [6], [41]. In the renewable resources, the resource may be used again for another job after being released [1]. The resource once used by some jobs, cannot be assigned to any other job in the non-renewable [42]. While in the doubly constrained resource, both classes are used at the same time [43]. The additional resources are also divided into discrete resources in which the number of resources needed via a job is a positive integer [6], [11], [41] and continuous resources where the number of resources required for the job is priori unknown in given intervals [6], [41], [44]. The additional resources that involve the processing of jobs can be classified into processing resources where the resources are required exactly at the time of the job processing, and input-output resources where the resource is required either before the job processing or after [45]. Assigning jobs to machines can be categorized into unspecified and specified (i.e., pre-specified) [33] whereby the jobs are not preassigned to any machine and vice versa for specified.

In general, this research work only deals with unspecified unrelated parallel machine scheduling problems with additional recourse. Four resources are selected in the present study. These resources are processing resources, dynamic, discrete, and renewable. These resources are selected as they are the most frequently used and they reflect the real-world scheduling problems [1], [8], [11], [29], [31], [33], [35]-[37].

2.1. Notations and decision variables

The UPMR problem treats with several types of input data. These include a list of m available machines; a list of j jobs to be processed; R_{max} units of a certain resource; r_{jm} units of the resource and p_{jm} units of time, which needed to process job j at machine m. Notations:

- M: number of machines (indexed as m), where m=1,…., M.
- J: number of jobs that need to be processed (indexed as j), where j=1,...., J.
- T: the number of time periods appearing in the scheduling horizon (indexed as t), where t = 1,...., T.
- R_{max}: maximum allowed number of resources
- p_{jm}: processing time of job j on machine m.
- r_{jm}: resources needed to process job j on machine m.

Decision variables:

- x_{jmt}: 1 if job j ends its processing on machine m at time t, 0 otherwise.
- C_{max}: the maximum completion time of all jobs (makespan).

2.2. UPMR formal mathematical model

UPMR aim is to minimize the maximum completion time of the job makespan, C_{max}. This is referred to as the objective function.

$$C_{max} = max_{j=1,\dots,J} \sum_{m=1}^{M} \sum_{t=p_{jm}}^{T} t x_{jmt} , \forall j = 1, \dots, J \quad (1)$$

The following hard constraints must be satisfied:

H1. One machine must process exactly one job and the processing ends just at one time.

$$\sum_{m=1}^{M} \sum_{t=p_{jm}}^{T} x_{jmt} = 1 \ , \forall\, j = 1, \dots, J \quad (2)$$

H2. Jobs are not processed together at the same time in the same machine.

$$\sum_{j=1}^{J} \sum_{s=t}^{t+p_{jm}-1} x_{jms} \leq 1 \, , \forall m = 1, . , M; \ \forall t = 1, \dots, T \quad (3)$$

H3. Do not exceed the R_{max} units of resource at any time.

$$\sum_{j=1}^{J} \sum_{m=1}^{M} \sum_{s=t}^{t+p_{jm}-1} r_{jm} x_{jms} \leq R_{max}, \forall t = 1, \dots, T \quad (4)$$

For the sake of understanding the UPMR problem, an explanatory example has been applied on dataset 12×6 (1_JobCorre_R_inter). Table 2 embodies the processing time (p_{jm}) and resource consumption (r_{jm}) for that dataset. Each dataset contains its own specific matrix of p_{jm} and r_{jm} that are different from those existing in dataset 12×6. Table 3 and Figure 1 represent the result of makespan C_{max} that was obtained based on (1) and taking into account the application of the constraints indicated in (2), (3) and (4).

Table 2. p_{jm} and r_{jm} (in brackets) of the dataset 12×6

Jobs	Machines					
	1	2	3	4	5	6
1	16 (2)	13 (4)	23 (1)	15 (1)	21 (3)	11 (1)
2	57 (2)	73 (6)	63 (7)	72 (4)	56 (5)	59 (2)
3	21 (5)	19 (1)	15 (2)	4 (4)	6 (3)	19 (2)
4	74 (9)	73 (9)	60 (6)	57 (2)	68 (8)	68 (4)
5	113 (9)	99 (9)	106 (9)	99 (9)	105 (9)	101 (9)
6	61 (3)	64 (3)	65 (6)	56 (5)	65 (3)	56 (8)
7	72 (3)	62 (3)	63 (8)	79 (7)	69 (7)	70 (4)
8	31 (2)	22 (4)	25 (4)	20 (5)	35 (4)	26 (2)
9	97 (8)	100 (8)	99 (9)	99 (8)	100 (7)	86 (8)
10	61 (6)	62 (7)	56 (3)	56 (5)	52 (7)	67 (7)
11	94 (5)	99 (9)	89 (9)	93 (9)	92 (8)	90 (5)
12	83 (5)	85 (5)	90 (9)	73 (5)	87 (9)	74 (3)

Table 3. The result obtained (c_{max}) = 145 on the dataset 12×6

Job	Start time	Finish time
11	0	94
6	0	64
3	0	15
4	0	57
8	0	35
9	0	86
10	15	71
5	35	140
1	57	72
7	64	126
12	72	145
2	86	145

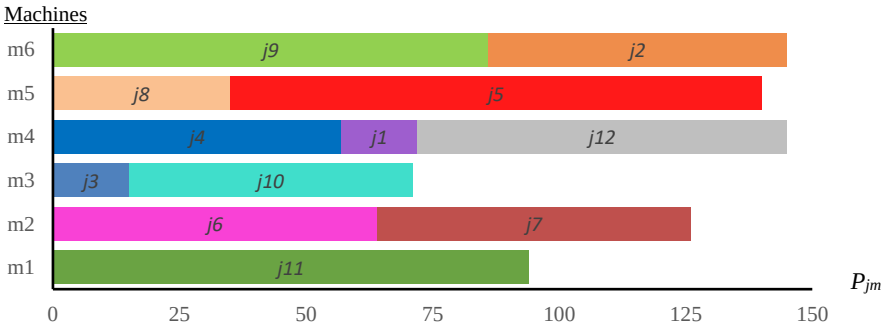

Figure 1. Makespan of dataset 12×6

3. PROPOSED ALGORITHM

GA has been implemented to solve many optimization problems and show the ability to produce quality solutions [46], [47]. It imitates the biological evolution process of chromosomes through the use of selection, crossover, and mutation operators. Chromosomes represent the solutions to problems and these solutions are evaluated in terms of their fitness value. Basically, GA begins from a group of solutions represented by chromosomes called population as shown in Figure 2, which may be a stochastic generator [40]. A chromosome is defined as a group of genes. Each gene, in any group, is characterized by having a unit of information on the problem solution. The generation of the new solution is carried out in eight steps as:

Step 1: The parameters are initialized.
Step 2: The chromosome representation is encoded for a problem.
Step 3: The initial population is produced randomly.
Step 4: The objective function is applied to evaluate the fitness value of each chromosome in the population.
Step 5: The selection process of chromosomes in the population is applied according to the type of selection, where each chromosome has a fitness value.
Step 6: The crossover process is carried out to create new population chromosomes by selecting parents and offspring based on the crossover probability rate.
Step 7: Mutation process were implemented on the new chromosomes based on the mutation probability rate.
Step 8: Feasibility of all chromosomes were checked based on the compliance of the constraints, otherwise the repair mechanism is executed.
Step 9: Step 4 is repeated until an optimal solution is achieved or until the stop criterion is satisfied.

3.1. Chromosome representation

A chromosome is a group of genes that contains an information on solution to the problem to be solved as indicated in Figure 3. As illustrated in the problem formulation, two important pieces of information are provided for UPMR problem: the first information is the machines and jobs, and the second information is the time periods and resource allocation that are needed for processing the jobs in any possible assignment. In this problem, the permutation encoding is utilized as it processes numbers only.

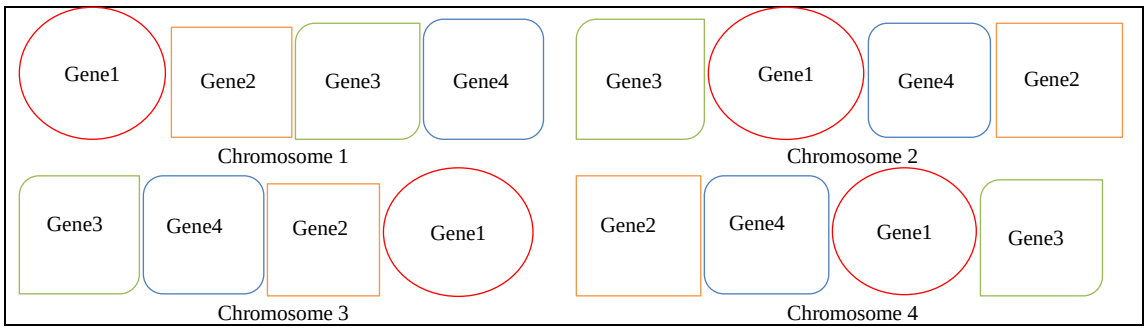

Figure 2. Population of solutions

Job sequence J	j_2	j_1	j_3	j_4	j_5
Machine assignment M	m_1	m_2	m_1	m_2	m_1

Figure 3. Chromosome representation

3.2. Initial population

It is defined as the starting GA population that contains feasible solutions. In general, the initial population is generated randomly and must satisfy all constraints. Figure 4 shows the pseudocode to generate the initial population. Though a large number of researchers used the single population, quite a few researchers use the multi-population recently [48], [49]. Multi-population makes the genetic algorithm allow the search space of the problem to be explored by one or more populations, while others can also perform exploitation in the same space (see Figure 5). It is expected that, by incorporating both exploitation and exploration, the quality of the solution would increase.

```
Begin
    Generate the initial solution.
    J number of jobs
    M number of machines
    Read the p_jm & r_jm & R_max
    For each chromosome in the population
        For each job that not selected yet
            Select one job randomly and assign it to any feasible machine
        End For
    End For
    Add the generated solution in the population
End
```

Figure 4. Initial population initialization

3.3. Selection operator

In the selection process, the appropriate chromosomes are chosen from the population to create a new population for the next iteration (until the stopping condition is met). In this study, the roulette wheel selection and K-tournament selection will be used to select the chromosomes because they are the most common and used in genetic algorithm [50], [51].

3.3.1. Roulette wheel selection

In roulette wheel selection, the chromosomes with the highest fitness value have a greater opportunity to be selected [52], [53]. The probability for each chromosome is calculated using (5):

$$p_c = \frac{f_c}{\sum_{c=1}^{n} f_c} \quad (5)$$

f_c indicates the fitness value of each chromosome. Fitness value means that the chromosome, which has less value of C_{max}, considering the best solution and its fitness value, is higher. As for the chromosomes, the

higher C_{max} value is, the lower fitness value is. While n is the number of chromosomes in the population. $\sum_{c=1}^{n} f_c$ is the total fitness of all chromosomes in the population. Figure 6 shows the fitness value and the fitness probability for each chromosome existing in population. Then, a random number is generated between 0 and 1. This number is spun like a roulette wheel to selected the chromosome.

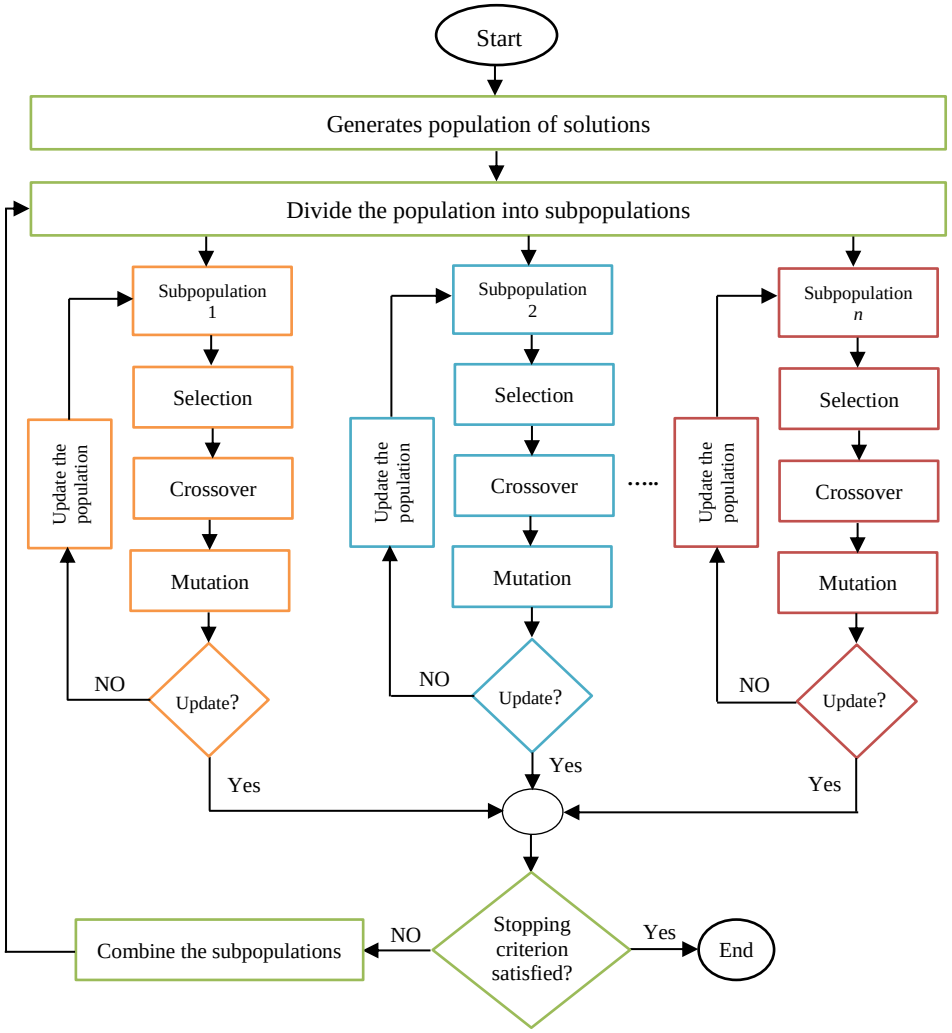

Figure 5. Flowchart of multi-population

Chromosome, c_n	c_1	c_2	c_3	c_4	c_5
Fitness value	1.5	1	2.5	2	3
Fitness probability, p_n	0.15	0.10	0.25	0.20	0.30

Figure 6. Fitness value and probability in a chromosome

3.3.2. *K*-tournament selection

The chromosomes are selected based on selecting the best solution among *K* chosen solutions from the population. In this approach, there is a possibility of selecting no-good solutions, however, it has more capability of avoiding local optima. The selection of value K is optional (for example: K=0.3 N) and refers to the competing chromosomes that will be randomly selected out of their total number. For instance, if the population have 30 solutions, then the *K* is 9. After that, the solution that has the best fitness value (less C_{max}) is selected among those 9.

3.4. Crossover operator

GA combines two chromosomes (parents) to generate new chromosomes (referred to as offspring's). The major goal of crossover is to produce a new chromosome, hoping it to be higher in status and rank than their parents (take the good features and attributes of each parent). In this study, one-point and uniform crossover will be used to combine two chromosomes [51], [54]. These two types were used because they are the most common and because they are completely different. The crossover point is generated randomly to involve the range of values from 0.2 to 0.8 and these values change for each iteration.

3.4.1. One-point crossover

One choice only is available at the crossover point (randomly). Genes to the right (or left) of that point are swapped between the two parent chromosomes to produce two offspring chromosomes (children) as illustrates in Figure 7. Basically, the offspring is created by combining chromosome1 (i.e., parent) starting from the 1st gene until the crossover point. Next, the remaining genes were taken from chromosome2.

```
Input z is a population, cr crossover rate
Begin
    n ← the number of chromosomes in z
    cn ← n * cr    // Chromosomes on which the crossover process will be applied
    For i=1 to cn/2
        cp ← random crossover point, 0.2 ≤ cp ≤ 0.8
        x ← select chromosome from z
        y ← select chromosome from z
        ng ← the number of genes in x or y
        cpn ← ng * cp
        For j=1 to cpn
            Swap x_i with y_i
        End For
        Apply repair mechanism for x, y
    End For
End
```

Figure 7. One-point crossover pseudocode

3.4.2. Uniform crossover

Uniform crossover produces offspring in a different method to point crossover methods. Crossover points are not selected but rather uniform crossover simply considers each bit position of the two parents. In contrast to one-point crossover the chromosomes are combined at the gene level in a uniform crossover. The genes taken from both chromosomes (parents) are selected randomly to generate chromosomes (offspring) based on crossover point as shown in Figure 8.

```
Input z is a population, cr crossover rate
Begin
    n ← the number of chromosomes in z
    cn ← n * cr        // Chromosomes on which the crossover process will be applied
    For i=1 to cn/2
        cp ← random crossover point, 0.2 ≤ cp ≤ 0.8
        x ← select chromosome from z
        y ← select chromosome from z
        ng ← the number of genes in x or y
        cpn ← ng * cp
        For j=1 to cpn
            rg ← random number, 1 ≤ rg ≤ ng
            Swap x_rg with y_rg
        End for
        Apply repair mechanism for x, y
    End for
End
```

Figure 8. Uniform crossover pseudocode

3.5. Mutation operator

Producing an offspring through the use of the crossover operator alone makes the GA stuck in the local optima, causing, thus, the survival of the good parts of the parents in each generation (i.e., the frequent transferring of the same genes from one generation to another). This problem is mitigated using the mutation operator through proving new offspring that is different from parents, and encouraging and motivating diversity in the population [55]. The insertion and inversion mutation will be applied in this study. Insertion mutation selects a gene at random and inserts it in a random position as listed in Figure 9. Whereas in inversion mutation, two positions within the same chromosome are randomly selected and then the sub-genes between these two positions are inverted as illustrated in Figure 10.

```
Input x is a population, mr mutation rate
Begin
    n ← the number of chromosomes in x
    mn ← n * mr        // Chromosomes on which the mutation process will be applied
    For i=1 to mn
        rx ← random chromosome, 1 ≤ rx ≤ n
        c ← select chromosome (rx indexed) from x
        ng ← number of genes (job / machine) in chromosome c
        rgp ← random gene position, 1 ≤ rgp ≤ ng
        rgpn ← random gene for new position, 1 ≤ rgpn ≤ ng
        Move the gene rgp indexed to the rgpn position in
        chromosome c
        Apply repair mechanism
    End For
End
```

Figure 9. Insertion mutation pseudocode

```
Input x is a population, mr mutation rate
Begin
    n ← the number of chromosomes in x
    mn ← n * mr        // Chromosomes on which the mutation process will be applied
    For i=1 to mn
        rx ← random chromosome, 1 ≤ rx ≤ n
        rb ← random block of genes (job / machine) in chromosome rx
        reverse genes order in block rb
        Apply repair mechanism
    End For
End
```

Figure 10. Inversion mutation pseudocode

3.6. Repair mechanism

A repetition error in the job sequence (i.e., chromosome list) after the crossover or mutation operation might occur. The error includes assigning jobs to more than one machine (i.e., duplicate job), or job not being assigned to any machine (lost job). This causes a violation to the hard constraint (i.e., H1 – H3). Hence, a repair mechanism is used to guarantee that all constraints are being satisfied. The repair mechanism pseudocode is shown in Figure 11. Besides, this mechanism is used to reorder one or more genes, and to reallocate a job to another machine to avoid the conflict.

```
Input x is a chromosome
Begin
    Remove all duplicated jobs in x
    Assign lost jobs on machines in x
    While conflict resources in x
        Reassign jobs on machines in x
    End While
End
```

Figure 11. Repair mechanism pseudocode

3.7. Parameters tuning and operators to solve UPMR

This section describes the proposed algorithms in solving UPMR. SGA begins with the parameter initialization based on the findings of the parameter tuning experiments and the initial population generation as shown in Figure 12. Then, the chromosomes in the population are evaluated to determine their fitness value. The next step involves creating a new chromosome (i.e., offspring).

The importance of GA parameters has been neglected and unexploited for an optimal solution [56]. The GA parameters very much influence and give a considerable impact on the solution quality [57]. Choosing the appropriate parameters value for GA is a significant task. Mainly, there are four essential parameters for the GA operation: crossover rate, mutation rate, population size, and number of iterations [14], [58], [59]. According to [14], [59], [60] the crossover and mutation rate is defined as the number of occurrences of crossover and mutation operation in an iteration respectively. A careful selection of crossover and/or mutation rates has been known to be significant to the success of genetic algorithms. In fact, it is also crucial to the success of the earlier problems, and even for different stages of the genetic process in a problem [61], [62]. The population size determines the total number of chromosomes. The population size is very sensitive in which a small population size would discourage exploration and easily trap in the local optima. However, for a large population size, it encourages exploration but GA would suffer from a high computational load [63]. Finally, the number of iterations is defined as the number of cycles before the algorithm stops (or terminated). The number of iterations depends on the problem complexity in which it may require a large number of iterations compared to less complex problems. According to the discussion above, this work investigates the suitable parameter values for GA in solving UPMR as shown in Table 3. In this investigation, the best parameter values for SGA in solving UPMR problem based on an experimental test are illustrated in Table 4.

Table 3. The parameter value of the GA

Crossover rate	Mutation rate	Population size	Iteration no.
0.3	0.01	20	500
0.5	0.05	40	1000
0.7	0.1	60	2000
0.9	0.5	100	4000
			6000

Table 4. Exact values of parameters in GGA

Parameter	Value
Population size	40
Crossover rate	0.7
Mutation rate	0.1
Iteration no.	4000

Although that the parameters values are very impotent for the GA performance, the appropriate selection of the GA operators is also very essential to make balance between exploration and exploitation, escape from the local optima and get good quality solutions [14], [64]. Moreover, investigations on the best SGA operator were also conducted by experimenting with different combinations of operators tested. These operators are single vs multi population, roulette wheel vs k-tournament selection, one-point vs uniform crossover and insertion vs inversion mutation. The experiment also reveals that the combination of multi-population, k-tournament selection, uniform crossover and insertion mutation produces the best result.

All the metaheuristic algorithms are designed to search for the optimal solutions by following a specific procedure. Most of them uses the classic procedure which is inspired either from the natural phenomena such as the Hill climbing algorithm or inspired from evolutionary procedures such as GA. For some optimization problems, the solver algorithms must consider the problem information in order to improve the search capabilities. One of the recent procedures is the guided procedure which depends a prior knowledge about the undertaken problem. In this paper, guided approach applied in the crossover operator to improve the SGA performance. The same operators and parameters, that were used in SGA, are used in GGA except for the crossover process. In the uniform crossover, the swap between genes (job/machine) for the two parent's chromosomes is randomly carried out. Therefore, the result may lead to infeasible solutions that require repair mechanism to obtain feasible solutions. While the crossover in GGA, the swap between genes is heuristically carried out. As such, this swap will be carried out if there is no chromosome conflict (the constraints are satisfied). Consequently, this type of swap will not need the repair mechanism.

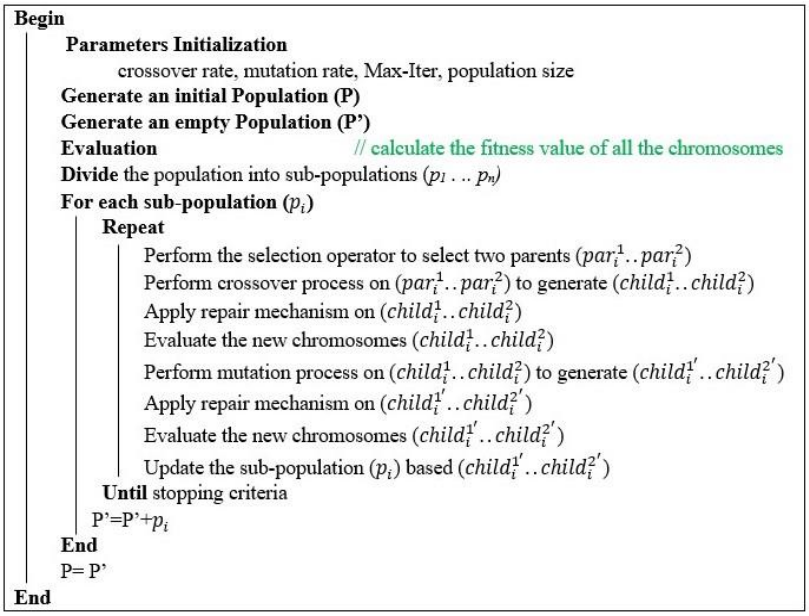

Figure 12. Pseudocode for the SGA

4. RESULTS AND DISCUSSION

The result of each benchmark instance reported by SGA and GGA is presented using the dataset as in [1], [8]. These instances can be found at http://soa.iti.es. The performances of the two variants of the proposed algorithm are evaluated in terms of the relative percentage deviation (RPD), average relative percentage deviation (ARPD), and in terms of average computational time (AvTime). The results derived from the SGA and GGA are compared to each other as well as to the work presented by [1]. RPD is computed using (6):

$$RPD = \frac{Heu_{sol} - LB}{LB} \times 100 \quad (7)$$

Heu_{sol} is the objective function value (i.e., *Makespan*) from the experiment and *LB* is the maximum lower bound based on [1]. *LB* is also called the best-known solution (BKS). The proposed algorithms were coded in C# 2012.5 and run on a PC with the components CPU Intel(R), Core (TM) i5, speed 2.20 GHz, and RAM 8.00 GB.

Table 5, Table 6 and Table 7 show the result of the investigation and comparison with other result found in the literature. The first column listed the dataset instances and the first row is the methods to compare (i.e., UPMR-P, JMR-P, M4, M5, ESS, EIG, SGA, and GGA indicated in columns show the results found as RPD). The penultimate row shows the ARPD and the last row illustrates the AvTime for each algorithm.

In general, the results indicate that GGA is able to produce better solutions in all instances when compared to SGA in terms of solution quality (RPD) and computational time (AvTime). In the small dataset (see Table 5), it is clear that the UPMR-P and JMR-P produce the best solution followed by EIG, ESS, M4, M5, GGA and SGA in terms of RPD. Here, GGA is able to produce solutions better than SGA which shows that the selected GA operators and parameters tuning give a significant improvement. Further, Table 5 shows that GGA outperforms other methods in terms of AvTime even though being a population-based algorithm.

As for the medium dataset (see Table 6), it becomes apparent that the best solution is produced, in terms of RPD, by the EIG and then by ESS, M4, M5, GGA, SGA, JMR-P and UPMR-P. The GGA indicates that it produces better solutions than the UPMR-P, JMR-P as well as SGA. Table 6 also shows that GGA outperforms other methods in terms of AvTime even though being a population-based algorithm.

In the large dataset, the UPMR-P and JMR-P were not used to solve the UPMR problem because it takes a large amount of time to solve this problem due to the complexity of this problem. Similarly, the EIG, ESS, M5 and M4 algorithm overcome GGA but they also underperform in terms of AvTime as shown in Table 7. On the other hand, the GGA outperforms SGA in all instances. Besides, GGA, as seen in Table 7, also outperforms other methods in terms of *AvTime* even though being a population-based algorithm.

Table 5. RPD, ARPD and AvTime for small instances

Instances	UPMR-P [1]	JMR-P [1]	M4 [8]	M5 [8]	ESS [37]	EIG [37]	SGA	GGA
8x2	0	0	0.10	0.24	0	0	2.21	1.40
8x4	0	1.94	0.73	1.17	0.66	0.53	2.72	1.54
8x6	0	1.88	1.06	0.96	0.13	0.04	2.63	1.98
12x2	0	0	0.59	0.67	0.35	0.40	2.58	1.46
12x4	1.14	2.48	1.87	2.16	1.47	1.14	3.25	2.42
12x6	1.44	1.05	1.14	1.18	0.75	0.48	1.50	1.38
16x2	0.21	0.21	0.56	0.63	0.23	0.14	1.78	1.45
16x4	6.46	4.25	1.39	1.71	1.19	1.00	4.04	2.02
16x6	8.71	5.99	1.40	1.61	1.22	0.77	5.00	2.42
ARPD	2.00	1.98	0.98	1.15	0.67	0.50	2.86	1.79
AvTime	2208.34	1969.92	0.0449	0.03522	5.5	5.5	0.02915	0.02305

Table 6. RPD, ARPD and AvTime for medium instances

Instances	UPMR-P [1]	JMR-P [1]	M4 [8]	M5 [8]	ESS [37]	EIG [37]	SGA	GGA
20x2	0.81	0.81	0.97	1.18	0.49	0.43	2.81	2.07
20x4	12.54	9.60	1.27	1.11	0.80	0.70	3.67	2.19
20x6	14.38	9.44	1.10	1.05	0.50	0.56	3.22	1.95
25x2	3.65	3.65	0.56	0.60	0.26	0.15	2.74	2.25
25x4	18.82	13.30	0.89	0.95	0.60	0.59	4.23	2.37
25x6	23.77	18.24	1.10	1.13	0.48	0.53	6.07	2.36
30x2	10.29	10.29	0.84	1.00	0.49	0.28	3.50	1.15
30x4	27.26	20.59	0.50	0.64	0.28	0.17	4.27	2.00
30x6	59.60	28.99	0.92	0.64	0.32	0.24	4.93	2.26
ARPD	19.01	12.77	0.91	0.92	0.47	0.41	3.94	2.07
AvTime	3600	3600	0.1965	0.14656	14.5	14.5	0.10279	0.08749

Table 7. RPD, ARPD and AvTime for large instances

Instances	M4 [8]	M5 [8]	ESS [37]	EIG [37]	SGA	GGA
50x10	1.03	1.07	0.36	0.27	5.04	2.45
50x20	1.78	1.37	0.50	0.40	4.66	3.01
50x30	1.47	1.51	0.35	0.34	3.91	2.50
150x10	0.10	0.73	0.30	0.28	2.56	1.36
150x20	1.37	1.43	0.55	0.28	4.73	2.45
150x30	1.41	1.24	0.46	0.26	6.06	3.73
250x10	0.87	0.65	0.25	0.15	3.34	1.57
250x20	1.01	0.82	0.40	0.23	4.60	2.96
250x30	1.00	0.77	0.36	0.18	4.76	2.36
350x10	0.63	0.48	0.23	0.12	2.84	2.01
350x20	0.68	0.47	0.30	0.15	5.05	2.98
350x30	0.78	0.65	0.24	0.10	4.67	3.15
ARPD	1.01	0.93	0.36	0.23	4.35	2.54
AvTime	184.93	102.25	73.1	60.5	19.5388	18.6273

5. CONCLUSION AND FUTURE WORK

This work investigates the significance of GA in solving the UPMR problem. Here, a guided genetic algorithm based on a specific GA operator is proposed to manipulate GA into producing a quality solution (refer to GGA). To measure the performance of GGA, the results are compared with other reported results from the literature. The comparisons revealed that the GGA shows better performance against SGA for solving the UPMR problem. An accurate selection of GA operators and parameters help in producing quality results. Comparison with other related work revealed that the GGA is outperformed by some these methods. Without doubt, it is able to give a competitive result and even better result than some of the methods. It is worth noting that the GGA outperformed, in terms of computational time, all the methods addressed in this research, especially the exact methods (i.e., UPMR-P and JMR-P) which require an extensive computation time to solve the problems. For future research, a hybridization of GGA with other metaheuristics methods can be applied to increase the performance of GGA in solving the UPMR problem. A metaheuristic that is good in exploitation (single-based metaheuristic) is highly recommended. This would greatly complement GGA (i.e., GGA is excellent in terms of exploration).

ACKNOWLEDGEMENTS

We are grateful to Universiti Malaysia Pahang (UMP) for supporting the research project through University Postgraduate Research Grant Scheme (PGRS1903187).

REFERENCES

[1] L. F.-Peyro, F. Perea, and R. Ruiz, "Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources," *European Journal of Operational Research*, vol. 260, no. 2, pp. 482–493, Jul. 2017, doi: 10.1016/j.ejor.2017.01.002.

[2] Salem and A. Hassan, *Unrelated parallel machine scheduling with sequence-dependent setup times and machine eligibility restrictions for minimizing the makespan*. University of Central Florida, 1999.

[3] C. M. Joo and B. S. Kim, "Parallel machine scheduling problem with ready times, due times and sequence-dependent setup times using meta-heuristic algorithms," *Engineering Optimization*, vol. 44, no. 9, pp. 1021–1034, Sep. 2012, doi: 10.1080/0305215X.2011.628388.

[4] J. P. Arnaout, R. Musa, and G. Rabadi, "A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines - Part II: Enhancements and experimentations," *Journal of Intelligent Manufacturing*, vol. 25, no. 1, pp. 43–53, Feb. 2014, doi: 10.1007/s10845-012-0672-3.

[5] R. Gedik, C. Rainwater, H. Nachtmann, and E. A. Pohl, "Analysis of a parallel machine scheduling problem with sequence dependent setup times and job availability intervals," *European Journal of Operational Research*, vol. 251, no. 2, pp. 640–650, Jun. 2016, doi: 10.1016/j.ejor.2015.11.020.

[6] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, "Scheduling under resource constraints," in *Handbook on scheduling: from theory to applications, New York: Springer: Verlag Berlin Heidelberg*, 2007, pp. 425–475.

[7] A. Bitar, S. D.-Pérès, C. Yugma, and R. Roussel, "A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing," *Journal of Scheduling*, vol. 19, no. 4, pp. 367–376, Aug. 2016, doi: 10.1007/s10951-014-0397-6.

[8] F. Villa, E. Vallada, and L. F.-Peyro, "Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource," *Expert Systems with Applications*, vol. 93, pp. 28–38, Mar. 2018, doi: 10.1016/j.eswa.2017.09.054.

[9] R. L. Daniels, S. Y. Hua, and S. Webster, "Heuristics for parallel-machine flexible-resource scheduling problems with unspecified job assignment," *Computers and Operations Research*, vol. 26, no. 2, pp. 143–155, 1999, doi: 10.1016/S0305-0548(98)00054-9.

[10] H. Kellerer, "An approximation algorithm for identical parallel machine scheduling with resource dependent processing times," *Operations Research Letters*, vol. 36, no. 2, pp. 157–159, Mar. 2008, doi: 10.1016/j.orl.2007.08.001.

[11] L. Fanjul, F. Perea, and R. Ruiz, "Algorithms for the unspecified unrelated parallel machine scheduling problem with additional resources," in *Proceedings of 2015 International Conference on Industrial Engineering and Systems Management, IEEE IESM 2015*, Oct. 2016, pp. 69–73, doi: 10.1109/IESM.2015.7380139.

[12] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, Sep. 2003, doi: 10.1145/937503.937505.

[13] Z. Michalewicz and D. B. Fogel, "Constraint-handling techniques," in *How to Solve It: Modern Heuristics, Berlin, Heidelberg: Springer Berlin Heidelberg*, 2004, pp. 231–270.

[14] E. G. Talbi, *Metaheuristics: From Design to Implementation*. Chichester: John Wiley & Sons and Sons Inc, 2009.

[15] G. Tian, Y. Ren, Y. Feng, M. C. Zhou, H. Zhang, and J. Tan, "Modeling and planning for dual-objective selective disassembly using and/or graph and discrete artificial bee colony," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2456–2468, Apr. 2019, doi: 10.1109/TII.2018.2884845.

[16] W. Wang *et al*., "Dual-objective program and improved artificial bee colony for the optimization of energy-conscious milling parameters subject to multiple constraints," *Journal of Cleaner Production*, vol. 245, p. 118714, Feb. 2020, doi: 10.1016/j.jclepro.2019.118714.

[17] F. Misni and L. S. Lee, "Harmony search for multi-depot vehicle routing problem," *Malaysian Journal of Mathematical Sciences*, vol. 13, no. 3, pp. 311–328, 2019.

[18] F. Wang, H. Zhang, and A. Zhou, "A particle swarm optimization algorithm for mixed-variable optimization problems," *Swarm and Evolutionary Computation*, vol. 60, p. 100808, Feb. 2021, doi: 10.1016/j.swevo.2020.100808.

[19] J. Liu, J. Yang, H. Liu, X. Tian, and M. Gao, "An improved ant colony algorithm for robot path planning," *Soft Computing*, vol. 21, no. 19, pp. 5829–5839, Oct. 2017, doi: 10.1007/s00500-016-2161-7.

[20] S. S. Chaudhry and W. Luo, "Application of genetic algorithms in production and operations management: A review," *International Journal of Production Research*, vol. 43, no. 19, pp. 4083–4101, Oct. 2005, doi: 10.1080/00207540500143199.

[21] M. H. Abed and A. Y. C. Tang, "Hybridizing genetic algorithm and record-to-record travel algorithm for solving uncapacitated examination timetabling problem," *Electronic Journal of Compuer Science and Information Technology (eJCSIT)*, vol. 4, no. 1, pp. 25–31, 2013.

[22] P. Sukhija, S. Behal, and P. Singh, "Face recognition system using genetic algorithm," *Procedia Computer Science*, vol. 85, pp. 410–417, 2016, doi: 10.1016/j.procs.2016.05.183.

[23] Y. Pratama, L. M. Ginting, E. H. L. Nainggolan, and A. E. Rismanda, "Face recognition for presence system by using residual networks-50 architecture," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 6, pp. 5488–5496, 2021, doi: 10.11591/ijece.v11i6.

[24] J. Xu, L. Pei, and R. Z. Zhu, "Application of a genetic algorithm with random crossover and dynamic mutation on the travelling salesman problem," *Procedia Computer Science*, vol. 131, pp. 937–945, 2018, doi: 10.1016/j.procs.2018.04.230.

[25] G. Ilewicz and A. Harlecki, "Multi-objective optimization and linear buckling of serial chain of a medical robot tool for soft tissue surgery," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 9, no. 1, p. 17, Mar. 2020, doi: 10.11591/ijra.v9i1.

[26] M. Afzalirad and M. Shafipour, "Design of an efficient genetic algorithm for resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions," *Journal of Intelligent Manufacturing*, vol. 29, no. 2, pp. 423–437, Feb. 2018, doi: 10.1007/s10845-015-1117-6.

[27] J. F. Chen, "Unrelated parallel machine scheduling with secondary resource constraints," *International Journal of Advanced Manufacturing Technology*, vol. 26, no. 3, pp. 285–292, Aug. 2005, doi: 10.1007/s00170-003-1622-1.

[28] A. Grigoriev, M. Sviridenko, and M. Uetz, "Unrelated parallel machine scheduling with resource dependent processing times," in *Lecture Notes in Computer Science*, vol. 3509, 2005, pp. 182–195.

[29] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Approximation algorithms for scheduling on multiple machines," in *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2005, vol. 2005, pp. 254–263, doi: 10.1109/SFCS.2005.21.

[30] J. F. Chen and T. H. Wu, "Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints," *Omega*, vol. 34, no. 1, pp. 81–89, Jan. 2006, doi: 10.1016/j.omega.2004.07.023.

[31] E. B. Edis and C. Oguz, "Parallel machine scheduling with additional resources: A lagrangian-based constraint programming approach," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6697 LNCS, 2011, pp. 92–98.
[32] E. B. Edis and I. Ozkarahan, "A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions," *Engineering Optimization*, vol. 43, no. 2, pp. 135–157, Feb. 2011, doi: 10.1080/03052151003759117.
[33] E. B. Edis and C. Oguz, "Parallel machine scheduling with flexible resources," *Computers and Industrial Engineering*, vol. 63, no. 2, pp. 433–447, Sep. 2012, doi: 10.1016/j.cie.2012.03.018.
[34] E. B. Edis and I. Ozkarahan, "Solution approaches for a real-life resource-constrained parallel machine scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 58, no. 9–12, pp. 1141–1153, Feb. 2012, doi: 10.1007/s00170-011-3454-8.
[35] K. Fleszar and K. S. Hindi, "Algorithms for the unrelated parallel machine scheduling problem with a resource constraint," *European Journal of Operational Research*, vol. 271, no. 3, pp. 839–848, Dec. 2018, doi: 10.1016/j.ejor.2018.05.056.
[36] T. Arbaoui and F. Yalaoui, "Solving the unrelated parallel machine scheduling problem with additional resources using constraint programming," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10752 LNAI, 2018, pp. 716–725.
[37] E. Vallada, F. Villa, and L. F.-Peyro, "Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem," *Computers and Operations Research*, vol. 111, pp. 415–424, Nov. 2019, doi: 10.1016/j.cor.2019.07.016.
[38] G. Moslemipour, T. S. Lee, and D. Rilling, "A review of intelligent approaches for designing dynamic and robust layouts in flexible manufacturing systems," *International Journal of Advanced Manufacturing Technology*, vol. 60, no. 1–4, pp. 11–27, Apr. 2012, doi: 10.1007/s00170-011-3614-x.
[39] A. S. Kholimi, A. Hamdani, and L. Husniah, "Automatic game world generation for platformer games using genetic algorithm," in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Oct. 2018, pp. 495–498, doi: 10.1109/EECSI.2018.8752741.
[40] J. H. Holland, "Adaptation in Natural and Artificial Systems," *Ann Arbor: University of Michigan Press*, 1992, doi: 10.7551/mitpress/1090.001.0001.
[41] R. Slowinski, "Two approaches to problems of resource allocation among project activities- a comparative study," *Journal of the Operational Research Society*, vol. 31, no. 8, pp. 711–723, Aug. 1980, doi: 10.1057/jors.1980.134.
[42] D. Shabtay and M. Kaspi, "Parallel machine scheduling with a convex resource consumption function," *European Journal of Operational Research*, vol. 173, no. 1, pp. 92–107, Aug. 2006, doi: 10.1016/j.ejor.2004.12.008.
[43] L. Özdamar and G. Ulusoy, "A local constraint-based analysis approach to project scheduling under general resource constraints," *European Journal of Operational Research*, vol. 79, no. 2, pp. 287–298, Dec. 1994, doi: 10.1016/0377-2217(94)90359-X.
[44] J. Y. T. Leung, *Handbook of scheduling: Algorithms, models, and performance analysis*. CRC press, 2004.
[45] J. Błażewicz, N. Brauner, and G. Finke, "Scheduling with discrete resource constraints," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis, E. J. Y-T.*, CRC press, 2004, pp. 23-1-23–18.
[46] L. Ma, J. Li, Q. Lin, M. Gong, C. A. Coello, and Z. Ming, "Cost-aware robust control of signed networks by using a memetic algorithm," *IEEE Transactions on Cybernetics*, vol. 50, no. 10, pp. 4430–4443, Oct. 2020, doi: 10.1109/TCYB.2019.2932996.
[47] G. Rivera, L. Cisneros, P. Sánchez-Solís, N. R.-Valdez, and J. R.-Osollo, "Genetic algorithm for scheduling optimization considering heterogeneous containers: A real-world case study," *Axioms*, vol. 9, no. 1, 2020, doi: 10.3390/axioms9010027.
[48] J. K. Cochran, S. M. Horng, and J. W. Fowler, "A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines," *Computers and Operations Research*, vol. 30, no. 7, pp. 1087–1102, 2003, doi: 10.1016/S0305-0548(02)00059-X.
[49] N. R. Sabar, A. Turky, M. Leenders, and A. Song, "Multi-population genetic algorithm for cardinality constrained portfolio selection problems," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10860 LNCS, 2018, pp. 129–140.
[50] K. Hingee and M. Hutter, "Equivalence of probabilistic tournament and polynomial ranking selection," *2008 IEEE Congress on Evolutionary Computation, CEC 2008*, 2008, pp. 564–571, doi: 10.1109/CEC.2008.4630852.
[51] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.
[52] N. Nuntasen and S. Innet, "Application of genetic algorithm for solving university timetabling problems: a case study of Thai universities," *UTCC Engineering Research Papers*, pp. 128–133, 2007, [Online]. Available: http://eprints.utcc.ac.th/990/
[53] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, Mar. 2012, doi: 10.1016/j.physa.2011.12.004.
[54] G. K. Soon, T. T. Guan, C. K. On, R. Alfred, and P. Anthony, "A comparison on the performance of crossover techniques in video game," in *Proceedings - 2013 IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2013*, Nov. 2013, pp. 493–498, doi: 10.1109/ICCSCE.2013.6720015.
[55] I. Korejo, S. Yang, K. Brohi, and K. Z.U.A, "Multi-population methods with adaptive mutation for multi-modal optimization problems," *International Journal on Soft Computing, Artificial Intelligence and Applications*, vol. 2, no. 2, pp. 1–19, 2013, doi: 10.5121/ijscai.2013.2201.
[56] Y. Zhang, J. Guo, and J. Zhou, "Dynamic mutation genetic algorithm," in *Dianzi Keji Daxue Xuebao/Journal of the University of Electronic Science and Technology of China*, 2002, vol. 31, no. 3, p. 234, doi: 10.1109/icsmc.1996.565436.
[57] A. L. Tuson and P. Ross, "Adapting operator probabilities in genetic algorithms," *Univeristy of Edinburgh*, 1995.
[58] A. Rexhepi, A. Maxhuni, and A. Dika, "Analysis of the impact of parameters values on the genetic algorithm for TSP," *International Journal of Computer Science Issues*, vol. 10, no. 1, pp. 158–164, 2013.
[59] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, "Choosing mutation and crossover ratios for genetic algorithms-a review with a new dynamic approach," *Information (Switzerland)*, vol. 10, no. 12, p. 390, Dec. 2019, doi: 10.3390/info10120390.
[60] K. A. De Jong and W. M. Spears, "A formal analysis of the role of multi-point crossover in genetic algorithms," *Annals of Mathematics and Artificial Intelligence*, vol. 5, no. 1, pp. 1–26, Mar. 1992, doi: 10.1007/BF01530777.
[61] W. Y. Lin, W. Y. Lee, and T. P. Hong, "Adapting crossover and mutation rates in genetic algorithms," *Journal of Information Science and Engineering*, vol. 19, no. 5, pp. 889–903, 2003.
[62] Y. A. Zhang, M. Sakamoto, and H. Furutani, "Effects of population size and mutation rate on results of genetic algorithm," in *Proceedings - 4th International Conference on Natural Computation, ICNC 2008*, vol. 1, 2008, pp. 70–75, doi: 10.1109/ICNC.2008.345.

[63] O. Roeva, S. Fidanova, and M. Paprzycki, "Influence of the population size on the genetic algorithm performance in case of cultivation process modelling," *2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013*, 2013, pp. 371–376
[64] E. Vallada and R. Ruiz, "A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times," *European Journal of Operational Research*, vol. 211, no. 3, pp. 612–622, Jun. 2011, doi: 10.1016/j.ejor.2011.01.011.

BIOGRAPHIES OF AUTHORS

Munther H. Abed received a B.S. degree in Computer engineering and information technology from University of Technology, Iraq in 2002, and an M.Sc. degree in Information technology from University Tenaga National, Malaysia in 2013. He is currently pursuing the Ph.D. degree in Information Technology, Faculty of Computing, College of Computing and Applied Sciences, UMP, Malaysia. He mainly interested in Metaheuristics, Hybrid algorithm, Timetabling, Scheduling, and Robotics. He can be contacted at email: munt1979@yahoo.com.

Mohd Nizam Mohmad Kahar received a PhD degree in Computer Science from the University of Nottingham, United Kingdom. He has been with Universiti Malaysia Pahang (UMP), Malaysia, where he is currently an Associate Professor in the Faculty of Computing. His research interests include solving real-world optimization problems such as the timetabling and routing problems using metaheuristics or nature-inspired algorithm. He can be contacted at email: mnizam@ump.edu.my.