

# Hybridizing guided genetic algorithm and single-based metaheuristics to solve unrelated parallel machine scheduling problem with scarce 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 Dec 18, 2021

Revised Sep 30, 2022

Accepted Oct 15, 2022

### Keywords:

Genetic algorithm

Great deluge

Hybridization

Makespan

Tabu search

Unrelated parallel machine scheduling with resources

Variable neighborhood search

## ABSTRACT

This paper focuses on solving unrelated parallel machine scheduling with resource constraints (UPMR). There are  $j$  jobs, and each job needs to be processed on one of the machines aim at minimizing the *makespan*. Besides the dependence of the machine, the processing time of any job depends on the usage of a rare renewable resource. A certain number of those resources ( $R_{max}$ ) can be disseminated to jobs for the purpose of processing them at any time, and each job  $j$  needs units of resources ( $r_{jm}$ ) when processing in machine  $m$ . When more resources are assigned to a job, the job processing time minimizes. However, the number of resources available is limited, and this makes the problem difficult to solve for a good quality solution. Genetic algorithm shows promising results in solving UPMR. However, genetic algorithm suffers from premature convergence, which could hinder the resulting quality. Therefore, the work hybridizes guided genetic algorithm (GGA) with a single-based metaheuristics (SBHs) to handle the premature convergence in the genetic algorithm with the aim to escape from the local optima and improve the solution quality further. The single-based metaheuristics replaces the mutation in the genetic algorithm. The evaluation of the algorithm performance was conducted through extensive experiments.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



## 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

At the age of technology advancement, the demand for fast and optimum production in manufacturing industries is increasing. These motivated decision-makers and researchers utilize the intelligent system for an optimum scheduling of the production process. This will increase the profits, decrease costs, whilst satisfy the customer needs. Scheduling of the production is deemed one of the considerable activities of a company when it comes to the operational level as it assists in keeping the company competitive in the demanding consumer markets. The company needs to effectively utilize its resources, meet production deadlines, reduce production costs and other constraints while fulfilling customer satisfaction. The relevance and potential of research and application in the manufacturing area are enormous and, this had attracted researchers to investigate problems in production scheduling from various perspectives over the previous years [1]. Of the crucial scheduling problems is the parallel machine scheduling problem (PMS). Researchers have classified the PMS problem as being non-deterministic polynomial-time hardness (NP-hard) even with the utilization of more than one machine (two machines), and they considered it as a combinatorial optimization problem [2]. Many methods were used and applied to provide a feasible solution to this problem, including exact method, approximation and heuristic

methods. Exact algorithms guarantee optimal solutions, because it explores all possible solutions in the entire search space and it would be ideal choice for a small size problem. However, for large size problems, no such algorithms have existed that able to solve in polynomial time [3]. Many works seen in the scientific literature used  $\rho$ -approximation approaches and problem-based heuristics to solve the large size optimization problems [4], [5]. These algorithms seek solutions which are near-optimal at a reasonable computation cost regardless to securing optimality and feasibility [6]. These methods are not suitable for a large variety of optimization problems because they are designed to handle a specific problem [7]. This paper proposed a hybridization of genetic algorithm and single-based metaheuristic algorithms in solving the unrelated parallel machine scheduling with resource (UPMR). The rest of the research is arranged where section 2 describes the parallel machine scheduling and section 3 deals with related work. A description of the problem, including the constraints, is discussed in section 4. Section 5 and section 6 describe the proposed algorithm and the results and discussion, respectively. Finally, section 7 presented the conclusion and future work.

## 2. PARALLEL MACHINE SCHEDULING

In this problem, the machines can be classified into five different classes depending on the machines nature which include single machine  $\emptyset$ , parallel dedicated machines  $PD$ , identical parallel machines  $P$ , uniform parallel machines  $Q$  and unrelated parallel machines  $R$  [8]. Single machine ( $\emptyset$ ) is the simplest of all possible machine environments and is a special case of all other more complicated machine environments [9]. As for Parallel dedicated machines, they are set of jobs that will be processed on each pre-determined machine [10]. Subsequently, the identical parallel machines mean that all of the machines have the same processing speed [11], and the uniform parallel machines mean that the machines have different speeds of execution but each machine works at a consistent rate [12]. Finally, the unrelated parallel machine,  $R$ , where the processing time of each job depends on the machine that it is assigned to. The unrelated parallel machine,  $R$  is much more complex and difficult compared to other models and closely resembles the real world problem in the industries [13]. The unrelated parallel machine can be divided into three types based on the constraints. These are classical unrelated parallel machine (UPM), unrelated parallel machine with sequence dependent setup time (UPMSP) and unrelated parallel machine with resources (UPMR).

### 2.1. Classical unrelated parallel machine (UPM)

In UPM, the processing of a number of jobs,  $j$  has to be performed on exactly one machine selected from a group of parallel machines. Many Jobs are now available to be processed at time zero and those jobs demand a processing time especially when jobs,  $j$  are distributed over to machines. The processing times of the jobs vary based on the machine for which jobs are assigned to. The classical parallel machines problem is an assignment problem that is typical in this context, and the only decision taken to solve the problem is which machine each job must be assigned to. Jobs that are assigned to machines can go into processing in any order until they are complete, and the machines, thus, are never left idle (even between jobs) [14]. This problem has been extensively explored in earlier literature for more than a decade [15].

### 2.2. Unrelated parallel machine with sequence dependent setup time (UPMSP)

In UPMSP, it includes a sequence of setup times and it is machine-dependent, where each machine has its own matrix of setup times, and these matrices differ from one another. The setup time on a machine between two jobs  $j$  and  $l$  differs from jobs  $l$  and  $j$  on the same machine. Additionally, the setup time between jobs  $j$  and  $l$  on one machine is different in other machines [16].

### 2.3. Unrelated parallel machine with resources (UPMR)

Recently, a new requirement emerges in which job,  $j$  requires, besides a machine,  $m$  a number of one or more extra resources,  $r$ . These additional resources could be considered human resources “machine operators”, automated *guided* vehicles, tools, pallets, fixtures, industrial robots or limited materials. These resources are deemed significant and have to be taken into account when assigning jobs to the machines. In addition, the number of resources that any job requires varies based on the machine a given job is assigned to. A sequence, on the other hand, means the computation of the beginning and end times of each job on the machines. Based on the availability of resources, idle times might be crucial to obtain a feasible solution. That has UPMR problem more complex when seen in terms of other problems. Due to the complexity and close similarity of UPMR to the real-world application [17], [18], this research's goal is to generate good quality solution in providing a solution to UPMR. UPMR problem exists in different manufacturing settings, such as car factories, food processing plants and many more [17].

### 3. RELATED WORK

This section described the approaches used for solving *unspecified dynamic* UPMR problem. Many works have been reported in the literature to solve this problem. For instance, [19] suggested a deterministic 3/2-approximation, 2-approximation and 4-approximation method to minimize the maximum completion time and weighted *completion* time. The 4-approximation method is superior to the other two methods. Grigoriev *et al.* [20] developed a 3.75-approximation algorithm using the rounding procedure to minimize the *makespan*. Thus, the results of this model outperform a deterministic 6.83-approximation and a randomized 4-approximation. A Lagrangian-based constraint programming (CP) method was proposed by [21] by relaxing the resource constraints. A comparison was carried out between the results of this method and the results of pure integer programming (IP) and pure CP models to uncover the fact or phenomenon that the method of the suggested Lagrangian-based CP yields very efficient and effective results. Edis and Oguz [22] proposed integer programming (IP) model, a relaxed IP based constraint programming (CP) method to solve the large size dataset and IP/CP model. The IP/CP model also outperform IP model and obtain near-optimal solutions for large size problems. Fanjul *et al.* [23] proposed two approaches: an integer linear programming (ILP) program and a two-phase approach based on solutions, named the fixing algorithm to minimize the *makespan*. The fixing algorithm outperforms the ILP program. Fanjul-Peyro *et al.* [17] formulate the problem via two integer linear programming problems mixed-integer linear programming (MILP). One of these methods relied on a model that was earlier presented by [22] and denoted by UPMR-S. The second one relies on the similarity to the problem of strip packing denoted by UPMR-P. They also presented three matheuristic strategies which included machine-assignment fixing (MAF), job-machine reduction (JMR), and greedy-based fixing (GBF) that were applied to each of these two models (UPMR-S and UPMR-P) and yielded MAF-S, JMR-S, GBF-S, MAF-P, JMR-P and GBF-P. The JMR-P approach outperform all approaches in most instances. Arbaoui and Yalaoui [18] presented CP model in order to minimize the  $C_{max}$ . Experimental results show the CP model outperform the exact and heuristic approaches in the literature (UPMR-S, UPMR-P, MAF-S and MAF-P) for both small and medium size instances. Two methods are also proposed by [24] to minimize the *makespan*. These methods are a MILP model and a CP model. The MILP performs much better than a pure CP model for large problem. Villa *et al.* [2] proposed Local search methods: Nawaz-Encore-Ham ( $NEH_{st}$ ), construction with swapping (SWA) and Nawaz-Encore-Ham ( $NEH_{res}$ ) and multi-pass heuristics (M1, M2, M3, M4 and M5) to minimize the *makespan*. Regarding small instances,  $NEH_{res}$  obtained the best results. While, in medium and large instances, multi-pass heuristics M4 and M5 obtained the best results, respectively. Vallada *et al.* [25] suggested four approaches scatter search (SS), enriched scatter search (ESS) and enriched iterated greedy (EIG) to minimize the *makespan*. The results obtained by EIG outperform the three methods for small, medium and large size instances. From the discussion, it shows that there is a need for a better algorithm in solving the UPMR for optimality. In the survey reported by [1], 10 metaheuristics are used where the genetic algorithm has been widely used. It stands for (50%) to solve many scheduling problems followed by simulated annealing which stands for (10%).

### 4. PROBLEM FORMULATION

The uniqueness of UPMR is the resources constraints, in which all jobs require resources when being processed in the machine. In view of the nature and usage of resources, they can be described by *classes*, *categories* and also *types* when allocating resources to machines [26]. In *classes*, the resources can be categorized into two, where it depends on “time” a resource requires by the jobs when being process by the machine [27]. Resources required during the processing of the jobs, is called *processing* resources [2], [17], [23]. However, if resources required either *before* or *after* processing of a job, then the resources are referred to as *input-output* resources [27], [28].

In *categories*, the resources can be divided into two [28], which include resource constraints and resource divisibility. Firstly, in the resource constraints categories, it includes resources that are renewable, nonrenewable and double constraint. A *renewable* resource allows the resources to be re-used after being released from other job [2], [17], [23]. However, for *nonrenewable* resources, the resources can only be used once by some job (i.e. cannot be assigned to any other job) [29]. As for double constraint, it uses both resources at the same time [28]. Second, in the resource divisibility, the resources are categories as *discrete* and *continuous*. In a *discrete* resource, the resources are to be assigned to jobs in discrete amounts from a limited finite group of possible allocations which may contain one element only [23]. However, in *continuous*, the resources can be assigned to jobs in random quantities from a certain interval [26].

In *type*, the resources can be divided into *static* and *dynamic* [18]. In *Static*, the allocation of resources to machines is fixed [30], while in the *dynamic*, the resources can be allocated and reallocated in view of the jobs' assignments [23], [24]. In terms of assigning jobs to the machines, this assignment can be of two types: *unspecified* and *specified* [28]. In the *unspecified*, the assignment of job to machine is not prefixed [23], while the assignment of job to machine is prefixed in the *specified* [31].

This work focuses on UPMR problem that is *unspecified, dynamic, processing resources, discrete and renewable*. These constraints are selected because they reflect the real-world problem [2], [17], [23], [27], [28]. This research focus on minimizing the *makespan* of the UPMR [32]. The following example sheds the light on the UPMR problem and raises the difference between this problem and the regular UPM. We need to take into mind the following example of an UPMR with two machines ( $m=2$ ), six jobs ( $j= 6$ ), 10 units of a scarce resource ( $R_{max}= 5 \times m$ ). The processing times  $p_{jm}$  and resource needs  $r_{jm}$  are  $j \times m$  matrices of the assignment machine  $m$  and job  $j$ , respectively.

$$p_{jm} = \begin{pmatrix} & j_1 & j_2 & j_3 & j_4 & j_5 & j_6 \\ m_1 & 15 & 32 & 26 & 41 & 10 & 29 \\ m_2 & 24 & 17 & 43 & 21 & 36 & 12 \end{pmatrix} \quad r_{jm} = \begin{pmatrix} & j_1 & j_2 & j_3 & j_4 & j_5 & j_6 \\ m_1 & 4 & 10 & 6 & 2 & 3 & 5 \\ m_2 & 4 & 2 & 1 & 9 & 5 & 7 \end{pmatrix}$$

As it is known, the difference between the UPM and UPMR problem is the resource constraints. Referring up to Figure 1, the UPM problem produces  $C_{max}$  51. However, this solution is infeasible where it violates the resource constraints in the UPMR problem. The maximum availability of resources is violated by two units, ( $r_{51}$  and  $r_{42}$ ) between time 0 and time 10, three units ( $r_{11}$  and  $r_{42}$ ) between time 10 and time 21 and three units ( $r_{31}$  and  $r_{62}$ ) between time 38 and time 50. To get a feasible solution from an infeasible solution while keeping the jobs assigned to the same machines, the idle-time is important. Figure 2 shows that resources are not overused but the result is poor quality and the *makespan* increased to  $C_{max}=84$ . It is possible to obtain feasible solutions without idle-time, but they are not optimal ( $C_{max}=77$ ) as shown in Figure 3. In Figure 4, resources are utilized, machines are optimized (always busy), and the *makespan* has not increased considerably when compared to the UPM problem which does not take into consideration the resource constraints shown in Figure 1 ( $C_{max}=72$ ).

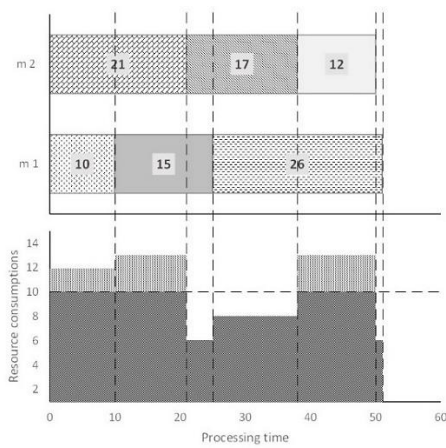


Figure 1. Infeasible solution

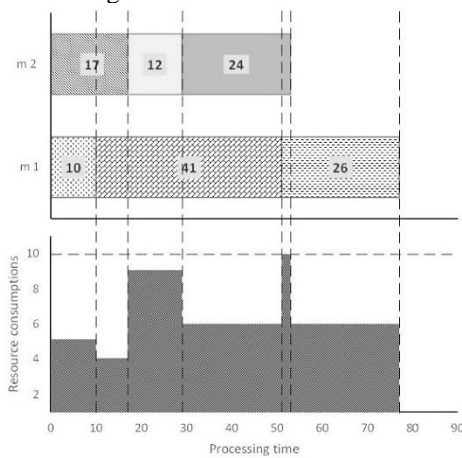


Figure 3. Feasible solution without idle-time

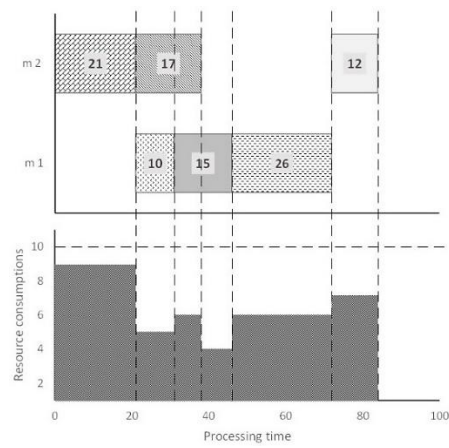


Figure 2. Feasible solution with idle-time

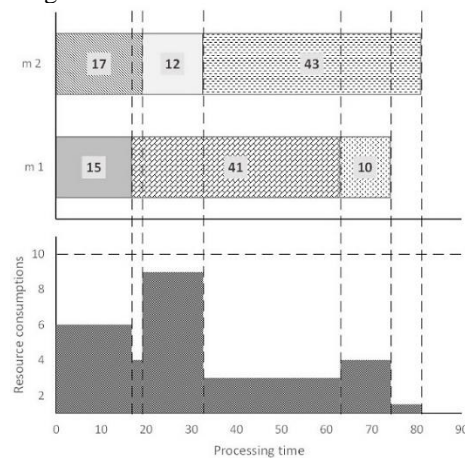


Figure 4. Near-optimal solution

#### 4.1. 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 by  $m$ ),  $m=1, \dots, M$ .
- $J$  number of jobs that requires processing (indexed by  $j$ ),  $j=1, \dots, J$ .
- $T$  number of the periods of time embodied in the scheduling horizon (indexed by  $t$ ),  $t = 1, \dots, T$ .
- $R_{max}$  maximum number of the permitted resources.
- $p_{jm}$  processing intervals of times of job  $j$  applied on machine  $m$ .
- $r_{jm}$  required number of the resources that are needed to process job  $j$  on machine  $m$ .

Decision variables:

- $x_{jmt}$  1 if job  $j$  accomplishes its processing period of time on machine  $m$  at time  $t$ , 0 otherwise.
- $C_{max}$  the maximum completion time of all jobs.

#### 4.2. Mathematical model

This section reveals the hard constraints and the objective function for the UPMR problem. It is taken into consideration that the hard constraint needs to be satisfied for a feasible solution as illustrated in (1) to (3). On the other hand, the objective function, that is widely applied in the UPMR, is to minimize the maximum completion time of the jobs (see (4)).

- Each job must be processed by exactly one machine; and finishes at exactly one time as shown in (1).

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

- One machine is allowed to process one job only at a time as this is illustrated in (2).

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

- Do not exceed the  $R_{max}$  units of the resource at any time. The formulation is shown in (3)

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

- The objective function is to minimize the *makespan* also known as ( $C_{max}$ ) and it is specified as in (4).

$$\text{Min } C_{max} = \max_{j=1, \dots, J} \sum_{m=1}^M \sum_{t=p_{jm}}^T t x_{jmt}, \quad \forall j = 1, \dots, J \quad (4)$$

### 5. PROPOSED ALGORITHM

As it is known, the genetic algorithm suffers from an imbalance between exploration and exploitation because its operation focuses more on exploring the search space rather than exploiting [33]. In addition, it also suffers from getting trapped into local optimal as it only takes the best solutions in population after the crossover and mutation operators. These drawbacks have not been solved even with many improvements to the standard genetic algorithm as reported in the literature. Such improvements included tuning parameters and selecting different types of operations [34]. Additionally, many researchers combine the genetic algorithm with single-based metaheuristic [35], [36].

At the beginning, the guided genetic algorithm (GGA) is applied with parameters tuning [37]. The parameters are examined by fixing one value of each parameter and changing the values of other parameters each time. This process is repeated with each value of the other parameters. On the other hand, the multi-population, guided crossover and insertion mutation are used. Next, the mutation process is replaced by the great deluge algorithm, tabu search algorithm and variable neighborhood search algorithm, to obtain guided genetic algorithm with great deluge (GGA-GD), guided genetic algorithm with tabu search (GGA-TS) and guided genetic algorithm with variable neighborhood search (GGA-VNS). The single-based metaheuristics are used to create a balance between exploration and exploitation and to address getting stuck in local optimal.

### 5.1. Guided genetic algorithm (GGA)

Two chromosomes are selected from the population, where the first represents the best one ( $BC$ ), while the second one is selected randomly based on k-tournament selection ( $RC$ ). Next, two unique opposite chromosomes for both chromosomes are generated, called  $\overline{BC}$  and  $\overline{RC}$ , based on (5) and (6) respectively.

$$\overline{BC} = Upper + Lower - CL \quad (5)$$

$$\overline{RC} = Upper + Lower - CL \quad (6)$$

Where  $Upper$  and  $Lower$  represent the boundaries of the problem (i.e., the highest and lowest value for the job) and  $CL$  represents the current location of the job. If the fitness value of the  $\overline{BC}$  and  $\overline{RC}$  are better than  $BC$  and  $RC$ , then, they replace them. Otherwise, mixed each chromosome with its own opposition chromosome together based on (7) and (8) respectively.

$$\overline{BC}_i = \begin{cases} BC_i & \text{if } ch_i < Cr \\ \overline{BC}_i & \text{otherwise} \end{cases} \quad (7)$$

$$\overline{RC}_i = \begin{cases} RC_i & \text{if } ch_i < Cr \\ \overline{RC}_i & \text{otherwise} \end{cases} \quad (8)$$

Where  $i$  represent an individual gene in each chromosome and  $Cr$  represent the crossover rate.  $ch$  denotes a chaotic series generated via Chaotic Tent map formula [38] (see (9)). The  $ch_0$  is generated randomly between 0 and 1. It is important to mention that the chaotic series are used to enhance the diversification of the crossover operator.

$$ch_{x+1} = \begin{cases} ch_x/0.7 & ch_x < 0.7 \\ 10/3(1 - ch_x) & ch_x \geq 0.7 \end{cases} \quad (9)$$

If the fitness value for the  $\overline{BC}_i$  and  $\overline{RC}_i$  are better than  $BC$  and  $RC$ , then  $BC$  and  $RC$  are replaced with the new generated one for the next generation. Otherwise, keep the  $BC$  and  $RC$  for the next generation.

### 5.2. Hybridizing guided genetic algorithm with great deluge algorithm

As mentioned, the great deluge algorithm will replace the mutation operator in the genetic algorithm. At each iteration, a neighbour solution is accepted if its quality is lower than the current value. Initially, the value of the level is set equal to the cost of the initial solution. Then, at each iteration, the level is decreased by the rain speed ( $UP$ ) using (10).

$$LEVEL = LEVEL - UP \quad (10)$$

This process will be repeated until the stopping criterion is reached (maximum number of iterations). The GD used the third type of the neighborhood structure based on the preliminary test as mentioned in sub-section 5.5. The pseudocode of hybridizing GGA-GD is presented in Figure 5.

### 5.3. Hybridizing guided genetic algorithm with tabu search algorithm

The procedure starts with a feasible initial solution obtained after the completion of the crossover operator of the genetic algorithm. This initial solution is stored both as the current and the best solution simultaneously. The algorithm will then form a set of neighboring solutions of the current solution that is not in the tabu list using a neighborhood structure. Then, the candidate solutions are evaluated using the objective function, and the best candidate solution which is not tabu is selected as the current solution. The fitness of the current solution will be compared to the best solution and, if it is better, will replace the best solution. This new current solution is added to tabu list and, if it is full, the oldest element is removed from the tabu list. Next, the whole procedures are repeated for a certain number of iterations [39]. Based on preliminary test, the TS used the fourth type of neighborhood structure (see sub-section 5.5). The pseudocode of GGA-TS is presented in Figure 6.



```

1. Begin
2. Parameters Initialization
3. Generate an initial Population (P)
4. Generate an empty Population (P')
5. Evaluation // calculate the fitness for each chromosome
6. Divide the population into sub-populations ( $p_1 \dots p_n$ )
7. For each sub-population ( $p_i$ )
8. Repeat
9.   Perform the selection operator to select two parents ( $par_i^1, par_i^2$ )
10.   Select the first chromosome that has the best fitness  $BC(par_i^1)$ 
11.   Select the second chromosome using K-tournament selection  $RC(par_i^2)$ 
12.   Perform crossover process on ( $par_i^1, par_i^2$ ) to generate ( $child_i^1, child_i^2$ )
13.   Find ( $\overline{par}_i^1, \overline{par}_i^2$ ) which are the opposite of ( $par_i^1, par_i^2$ ) // Using (5) and (6)
14.   If the fitness value of  $\overline{par}_i^1$  better than  $par_i^1$ , set  $child_i^1 = \overline{par}_i^1$ 
15.   Else mix  $\overline{par}_i^1$  with  $par_i^1$  to generate  $\overline{\overline{par}}_i^1$  // Using (7)
16.   If the fitness value of  $\overline{\overline{par}}_i^1$  better than  $par_i^1$ , set  $child_i^1 = \overline{\overline{par}}_i^1$ 
17.   Else set  $child_i^1 = par_i^1$ 
18.   If the fitness value of  $\overline{\overline{par}}_i^2$  better than  $par_i^2$ , set  $child_i^2 = \overline{\overline{par}}_i^2$ 
19.   Else mix  $\overline{\overline{par}}_i^2$  with  $par_i^2$  to generate  $\overline{\overline{par}}_i^2$  // Using (8)
20.   If the fitness value of  $\overline{\overline{par}}_i^2$  better than  $par_i^2$ , set  $child_i^2 = \overline{\overline{par}}_i^2$ 
21.   Else set  $child_i^2 = par_i^2$ 
22.   Evaluate the new chromosomes ( $child_i^1 \dots child_i^2$ )
23.    $BestChild_i$  =the best child ( $child_i^1$  OR  $child_i^2$ )

// Great deluge process
a.  $s = BestChild_i$ ;
b. Initiate the water level  $LEVEL = f(s)$ ;
c. Choose the rain speed  $UP$ ; // Using (11),  $UP > 0$ 
d. Repeat
e.   Generate a new neighbour  $s'$  of  $s$ ; //  $s' = N(s)$ 
f.   If  $f(s') < LEVEL$  Then  $s = s'$ ; // Accept the neighbour solution
g.    $LEVEL = LEVEL - UP$ ; // update the water level
h. Until Stopping criteria satisfied
i.  $LS\_Sol = s$ ;

24.   Evaluate the chromosomes generated by GD ( $LS\_Sol$ )
25.   Update the sub-population ( $p_i$ ) // replace the worst chromosome by  $LS\_Sol$ 
26. Until stopping criteria
27.    $P' = P' + p_i$ 
28. End
29.  $P = P'$ 
30. End

```

Figure 5. Pseudocode of GGA-GD

```

1. Begin
2. Parameters Initialization
3. Generate an initial Population (P)
4. Generate an empty Population (P')
5. Evaluation // calculate the fitness for each chromosome
6. Divide the population into sub-populations ( $p_1 \dots p_n$ )
7. For each sub-population ( $p_i$ )
8. Repeat
9.   Perform the selection operator to select two parents ( $par_i^1, par_i^2$ )
10.   Select the first chromosome that has the best fitness  $BC(par_i^1)$ 
11.   Select the second chromosome using K-tournament selection  $RC(par_i^2)$ 
12.   Perform crossover process on ( $par_i^1, par_i^2$ ) to generate ( $child_i^1, child_i^2$ )
13.   Find ( $\overline{par}_i^1, \overline{par}_i^2$ ) which are the opposite of ( $par_i^1, par_i^2$ ) // Using (5) and (6)
14.   If the fitness value of  $\overline{par}_i^1$  better than  $par_i^1$ , set  $child_i^1 = \overline{par}_i^1$ 
15.   Else mix  $\overline{par}_i^1$  with  $par_i^1$  to generate  $\overline{\overline{par}}_i^1$  // Using (7)
16.   If the fitness value of  $\overline{\overline{par}}_i^1$  better than  $par_i^1$ , set  $child_i^1 = \overline{\overline{par}}_i^1$ 
17.   Else set  $child_i^1 = par_i^1$ 
18.   If the fitness value of  $\overline{\overline{par}}_i^2$  better than  $par_i^2$ , set  $child_i^2 = \overline{\overline{par}}_i^2$ 
19.   Else mix  $\overline{\overline{par}}_i^2$  with  $par_i^2$  to generate  $\overline{\overline{par}}_i^2$  // Using (8)
20.   If the fitness value of  $\overline{\overline{par}}_i^2$  better than  $par_i^2$ , set  $child_i^2 = \overline{\overline{par}}_i^2$ 
21.   Else set  $child_i^2 = par_i^2$ 
22.   Evaluate the new chromosomes ( $child_i^1 \dots child_i^2$ )
23.    $BestChild_i$  =the best child ( $child_i^1$  OR  $child_i^2$ )

// Tabu search process
a. Parameters Initialization
b.  $s = BestChild_i$ ; // Initial solution
c. Initialize the tabu list;
d. Repeat
e.   Find best admissible neighbour  $s'$ ; // non tabu
f.    $s = s'$ ;
g.   Update tabu list;
h. Until Stopping criteria satisfied
i.  $LS\_Sol = s$ ;

24.   Evaluate the chromosomes generated by TS ( $LS\_Sol$ )
25.   Update the sub-population ( $p_i$ ) // replace the worst chromosome by  $LS\_Sol$ 
26. Until stopping criteria
27.    $P' = P' + p_i$ 
28. End
29.  $P = P'$ 
30. End

```

Figure 6. Pseudocode of GGA-TS

#### 5.4. Hybridizing guided genetic algorithm with variable neighborhood search algorithm

The variable neighborhood search algorithm (VNS) begins with a solution that is initial. After that, three steps constitute each iteration of the algorithm. These three steps are shaking, local search, and move as shown in Figure 7. VNS will form a set of neighboring solutions based on  $k^{\text{th}}$  neighborhood. After that, the candidate solutions are evaluated using the objective function, and a random candidate solution  $s'$  is shaken from  $N_k(s)$  to be the current solution. In this work, four types of neighborhood structures are used gradually as mentioned in sub-section 5.5. A procedure of local search known as hill climbing (HC) is applied to the solution  $s'$  to produce the solution  $s''$ . The present solution is changed by the new local optima  $s''$  if and only if there is a better solution that has been discovered (i.e.,  $f(s'') < f(s)$ ). The same search procedure, thus, is initiated from the solution  $s''$  in the first neighborhood  $N_1$ . HC process terminates when the maximum number of iterations stands for 3000 based on preliminary test. Concerning the neighborhood structure, the HC used the first type based on the preliminary test (see sub-section 5.5). If no better solution is located (i.e.,  $f(s'') \geq f(s)$ ), the algorithm moves to the next neighborhood  $N_{k+1}$ , randomly produces a new solution in this neighborhood, and aims to improve it.

```

1. Begin
2. Parameters Initialization
3. Generate an initial Population (P)
4. Generate an empty Population (P')
5. Evaluation // calculate the fitness for each chromosome
6. Divide the population into sub-populations ( $p_1 \dots p_n$ )
7. For each sub-population ( $p_i$ )
8.   Repeat
9.     Perform the selection operator to select two parents ( $par_i^1, par_i^2$ )
10.    Select the first chromosome that has the best fitness  $BC(par_i^1)$ 
11.    Select the second chromosome using K-tournament selection  $RC(par_i^2)$ 
12.    Perform crossover process on ( $par_i^1, par_i^2$ ) to generate ( $child_i^1, child_i^2$ )
13.    Find ( $\overline{par}_i^1, \overline{par}_i^2$ ) which are the opposite of ( $par_i^1, par_i^2$ ) // Using (5) and (6)
14.    If the fitness value of  $\overline{par}_i^1$  better than  $par_i^1$ , set  $child_i^1 = \overline{par}_i^1$ 
15.    Else mix  $\overline{par}_i^1$  with  $par_i^1$  to generate  $\overline{par}_i^1$  // Using (7)
16.    Else if the fitness value of  $\overline{par}_i^1$  better than  $par_i^1$ , set  $child_i^1 = \overline{par}_i^1$ 
17.    Else set  $child_i^1 = par_i^1$ 
18.    If the fitness value of  $\overline{par}_i^2$  better than  $par_i^2$ , set  $child_i^2 = \overline{par}_i^2$ 
19.    Else mix  $\overline{par}_i^2$  with  $par_i^2$  to generate  $\overline{par}_i^2$  // Using (8)
20.    Else if the fitness value of  $\overline{par}_i^2$  better than  $par_i^2$ , set  $child_i^2 = \overline{par}_i^2$ 
21.    Else set  $child_i^2 = par_i^2$ 
22.    Evaluate the new chromosomes ( $child_i^1, child_i^2$ )
23.     $BestChild_i =$ the best child ( $child_i^1$  OR  $child_i^2$ )

    // Variable neighbourhood search process
    a. Input: a set of neighbourhood structures  $N_k$  for  $k = 1, \dots, k_{\max}$  for shaking.
    b.  $s = BestChild_i$ ;
    c. Repeat
    d.    $k = 1$ ;
    e.   Repeat
    f.     Shaking: pick a random solution  $s'$  from the  $k^{\text{th}}$  neighbourhood  $N_k(s)$ ;
    g.      $s'' =$ hill climbing ( $s'$ ); // Generate a candidate neighbour
    h.     If  $f(s'') < f(s)$  Then
    i.        $s = s''$ ;
    j.       Continue to search with  $N_1$ ;  $k = 1$ ;
    k.     Otherwise,  $k = k + 1$ ;
    l.     Until  $k = k_{\max}$ 
    m. Until Stopping criteria
    n.    $LS\_Sol = s$ ;

24.   Evaluate the chromosomes generated by VNS ( $LS\_Sol$ )
25.   Update the sub-population ( $p_i$ ) // replace the worst chromosome by  $LS\_Sol$ 
26. Until stopping criteria
27.    $P' = P' + p_i$ 
28. End
29.  $P = P'$ 
30. End

```

Figure 7. Pseudocode of GGA-VNS

#### 5.5. Neighborhood strategies

The neighborhood structures have a prominent role in the performance of any SBH [33]. The existence of adequate neighborhood leads to enhance the ability of a SBH to generate good solutions [40]. In this work, the neighbor solution is created via four neighborhood structures:

- Select one job and insert it to different position within the same machine.
- Select one job and insert it to a different machine.
- Select two jobs from the same machine and swap their positions.
- Select two jobs from different machines and swap their positions.



## 5.6. Parameter tuning

The values of the parameters have direct effect on the performance of the metaheuristic algorithms [33]. Different problems and even different instances from the same problem data require different parameter values to reach an optimal or near optimal solution. The parameter settings for the proposed hybrid algorithms are discussed in this subsection. A preliminary test is conducted to determine the suitable parameters values. During the preliminary test, the genetic algorithm and SBH algorithms are executed 30 runs on ten instances (8×2, 12×4, 16×6, 20×2, 25×4, 30×6, 50×10, 150×20, 250×30 and 350×10) and the best results over 30 runs are reported. These instances are selected based on the size of the benchmark dataset. Table 1 summarizes the parameter settings of all the algorithms used in this study.

Table 1. The parameter settings of the hybridized algorithm

Parameter	Algorithm	Value
Population size	GGA	40
Crossover rate	GGA	0.7
Mutation rate	GGA	0.1
Number of iterations	GGA	4000
Number of iterations	GD	4000
Tabu list length	TS	40
Number of neighborhood solutions	TS	6
Number of iterations	TS	6000
Number of iterations	HC	3000
Number of iterations	VNS	3000

### 5.6.1. GA parameter settings

Genetic algorithm (GA) has four parameters, namely population size, crossover rate, mutation rate and number of generation (iteration). A preliminary test is used to determine the appropriate parameter values in guided genetic algorithm (GGA) [37]. Different values of the parameters are used and tested. The values used for population size are 20, 40, 60 and 100. In view of the results of the comparison carried out on all the datasets used, the most suitable value of the population size is 40 produces the best result obtained in seven out ten datasets. After setting the value of the population size, the crossover rate will be examined. The crossover rate must also be set. The values of the crossover rate that are tested are 0.3, 0.5, 0.7 and 0.9. The best crossover rate value for the eight datasets out of ten is 0.7. Next, the values of the mutation rate that are tested are 0.01, 0.05, 0.1 and 0.3. The value of the best mutation rate for the six datasets out of ten is 0.1. Finally, the values used for the number of iterations are 500, 1000, 2000, 4000 and 6000. Based on the results of the comparison carried out on all the datasets used, the most suitable value for the number of iterations is the value 4000 as the best results of eight datasets out of ten are obtained at this value.

### 5.6.2. GD parameter settings

GD is composed of two parameters. The first of these is rain speed ( $UP$ ), which requires to be tuning and is calculated using (11) [41]:

$$UP = \frac{f(\text{initial solution}) - BKS}{\text{number of iterations}} \quad (11)$$

$BKS$  refers to the best-known solution. The second parameter is the number of iterations, which is set to 4000 out of four values (2000, 3000, 4000, 5000 and 6000) based on preliminary test.

### 5.6.3. TS parameter settings

TS utilizes three parameters, the tabu list length  $TL$ , the number of neighborhood solutions  $N(s')$  and the number of iterations. The values used for tabu list length are 20, 30, 40 and 50. In addition, the values 4, 6, 8, and 10 are used for the number of neighborhood solutions. While, the number of iterations is represented by the values 2000, 3000, 4000, 5000 and 6000. Based on the preliminary test, two parameters were fixed to examine the third one and so on. The parameters value of the TS used in this study based on preliminary test are: the tabu list length is fixed to 40, the number of neighborhood solutions is fixed to 6 and the number of iterations is fixed to 4000.

### 5.6.4. HC and VNS parameter settings

HC and VNS contain only one parameter, the number of iterations. The setting of the parameter to 3000 for two algorithms is based on the results of the preliminary test. This preliminary test is, then, applied to the values of the different iterations (2000, 3000, 4000, 5000 and 6000).

## 6. RESULTS AND DISCUSSION

The proposed algorithms are evaluated and compared with state-of-the-art methods in terms of solution quality. The obtained results are reported as the relative percentage deviation (*RPD*) from the best-known solution (*BKS*). The best-known solution is the solution obtained by [2], [17], [25]. The relative percentage deviation *RPD* for each instance is computed using (12) [42].

$$RPD = \frac{Heu_{sol} - BKS}{BKS} \times 100 \quad (12)$$

*BKS* is the Best-Known Solution which can be optimal if the *makespan* is equal to the optimum *makespan* of the problem without resources (UPM). *Heu<sub>sol</sub>* is the best solution obtained by the proposed algorithm over all independent runs. The proposed algorithms are coded in C# 2012 and run on a PC with CPU Intel(R), Core (TM) i5, speed at 2.20 GHz and RAM 8.00 GB. A benchmark dataset used in this study is proposed by [2], [17], which consist of different sizes of dataset were used for the experiment (referred to as small, medium and large). The small size has 9 datasets, the medium has 9 datasets and the large has 12 datasets, where each data set has 50 instances. In total, there are 450 small instances, 450 medium instances and 600 large instances. The dataset can be reach at [43]. The number of resources is computed as  $R_{max} = 5 \times$  number of machines. The computational result of the given instance (8x2\_2\_U\_10\_100\_\_R\_inter) and  $C_{max}$  along with the processing time and resource consumption as shown in Table 2.

Table 2. The result obtained by GGA-VNS with  $C_{max} = 133$

Job	machine	Processing time $p_{jm}$		Resource consumption $r_{jm}$
		Start time	Finish time	
4	1	0	47	3
8	2	0	51	6
1	1	47	57	3
6	2	51	91	3
2	1	57	85	1
5	1	85	100	4
7	2	91	<b>133</b>	1
3	1	100	119	4

### 6.1. Small instances result

Our first evaluation is using small instances as shown in Table 3. GGA-VNS outperformed GGA-TS and GGA-GD by 56% in terms of *RPD*. When comparing our algorithms with the state-of-the-art methods, UPMR algorithm obtains the lowest *RPD* in 5 instances followed by the EIG method and JMR that obtain 4 and 2 instances respectively. The ESS, GGA-TS and GGA-VNS obtain 1 instance followed by M4, M5 and GGA-GD which didn't record any results that outperform the mentioned algorithms. In view of the average relative percentage deviation (*ARPD*), the EIG obtains the lowest (best) value followed by GGA-VNS, ESS, GGA-TS, GGA-GD, M4, M5, JMR and UPMR. Comparatively, in terms of the average time *AvTime*, the UPMR and JMR methods need approximately 2000 seconds on average, the ESS and EIG at around 5 seconds on average. The GGA-GD, M5, GGA-VNS, M4 and GGA-TS methods need less than 1 second on average.

Table 3. *RPD*, *ARPD* and *AvTime* for small instances

Instance	UPMR [17]	JMR [17]	M4 [2]	M5 [2]	ESS [25]	EIG [25]	GGA-GD	GGA-TS	GGA-VNS
8x2	<b>0</b>	<b>0</b>	0.10	0.24	<b>0</b>	<b>0</b>	0.90	0.46	0.29
8x4	<b>0</b>	1.94	0.73	1.17	0.66	0.53	0.59	0.73	0.46
8x6	<b>0</b>	1.88	1.06	0.96	0.13	0.04	0.72	0.44	0.47
12x2	<b>0</b>	<b>0</b>	0.59	0.67	0.35	0.40	0.93	0.84	0.61
12x4	<b>1.14</b>	2.48	1.87	2.16	1.47	<b>1.14</b>	1.07	1.31	1.19
12x6	1.44	1.05	1.14	1.18	0.75	0.48	0.93	<b>0.46</b>	0.67
16x2	0.21	0.21	0.56	0.63	0.23	<b>0.14</b>	0.51	0.34	0.19
16x4	6.46	4.25	1.39	1.71	1.19	<b>1.00</b>	1.04	1.40	1.15
16x6	8.71	5.99	1.40	1.61	1.22	0.77	0.96	1.29	<b>0.75</b>
<i>ARPD</i>	2.00	1.98	0.98	1.15	0.67	<b>0.50</b>	0.85	0.81	0.64
<i>AvTime</i>	2208.34	1969.92	0.0449	0.0352	5.5	5.5	<b>0.0346</b>	0.0451	0.0398

### 6.2. Medium instances result

Table 4 shows the results for medium instances. With respect to *RPD*, GGA-VNS and GGA-TS outperformed GGA-GD by 44% for each one (i.e., 4 instances for GGA-VNS and 4 instances for GGA-TS and

1 instance for GGA-GD). Taking *ARPD* into account, the GGA-VNS outperformed the GGA-TS and GGA-GD. In comparing the results of the study with the previous literature, the study found that EIG algorithm obtains the lowest *RPD* in 4 instances followed by the GGA-TS, ESS and GGA-VNS that obtain 3, 2 and 1 instances respectively. The UPMR, JMR, M4, M5 and GGA-GD did not record the lowest *RPD* compared to the earlier mentioned algorithms. In *ARPD*, the lowest (best) value is obtained by the EIG and then followed by ESS, GGA-VNS, GGA-TS, M4, M5, GGA-GD, JMR and UPMR. On the other hand, taking into consideration the average time *AvTime*, the study showed that the UPMR and JMR methods need 3600 seconds on average, whereas the ESS and EIG need around 14.5 seconds on average. In comparison, the GGA-GD, GGA-VNS, M5, GGA-TS and M4 methods show the need for less than 1 second on average.

Table 4. *RPD*, *ARPD* and *AvTime* for medium instances

Instance	UPMR [17]	JMR [17]	M4 [2]	M5 [2]	ESS [25]	EIG [25]	GGA-GD	GGA-TS	GGA-VNS
20x2	0.81	0.81	0.97	1.18	0.49	0.43	1.13	<b>0.42</b>	0.73
20x4	12.54	9.60	1.27	1.11	0.80	0.70	1.27	0.85	<b>0.69</b>
20x6	14.38	9.44	1.10	1.05	<b>0.50</b>	0.56	0.96	1.26	0.56
25x2	3.65	3.65	0.56	0.60	0.26	<b>0.15</b>	0.98	0.46	0.65
25x4	18.82	13.30	0.89	0.95	0.60	<b>0.59</b>	1.39	0.90	0.64
25x6	23.77	18.24	1.10	1.13	<b>0.48</b>	0.53	0.50	1.30	0.98
30x2	10.29	10.29	0.84	1.00	0.49	0.28	1.07	<b>0.25</b>	0.82
30x4	27.26	20.59	0.50	0.64	0.28	<b>0.17</b>	1.37	<b>0.17</b>	0.87
30x6	59.60	28.99	0.92	0.64	0.32	<b>0.24</b>	1.30	0.88	0.38
<i>ARPD</i>	19.01	12.77	0.91	0.92	0.47	<b>0.41</b>	1.11	0.72	0.70
<i>AvTime</i>	3600	3600	0.1965	0.1465	14.5	14.5	<b>0.1296</b>	0.1488	0.1335

### 6.3. Large instances result

Experiments of large instances are implemented as illustrated in Table 5. While GGA-VNS outperformed GGA-TS and GGA-GD by 75% (9 instances out of 12) in view of *RPD*. The GGA-VNS also outperformed the GGA-TS and GGA-GD in view of *ARPD*. In comparison with the state-of-the-art methods, EIG algorithm is found to obtain the lowest *RPD* in all instances. As far as *ARPD* is concerned, the study showed that the lowest (best) value is obtained by EIG and then followed by ESS, GGA-VNS, GGA-TS, M5, GGA-GD and M4. In respect of the average time *AvTime*, the GGA-GD, GGA-TS and GGA-VNS need around 19 seconds on average. Subsequently, the EIG, ESS, M5 and M4 need around 60 seconds, 73 seconds, 102 seconds and 184 seconds respectively.

Table 5. *RPD*, *ARPD* and *AvTime* for large instances

Instance	M4 [2]	M5 [2]	ESS [25]	EIG [25]	GGA-GD	GGA-TS	GGA-VNS
50x10	1.03	1.07	0.36	<b>0.27</b>	0.99	1.24	0.58
50x20	1.78	1.37	0.50	<b>0.40</b>	0.46	0.81	1.38
50x30	1.47	1.51	0.35	<b>0.34</b>	1.22	0.93	0.44
150x10	1.00	0.73	0.30	<b>0.28</b>	0.62	0.94	0.36
150x20	1.37	1.43	0.55	<b>0.28</b>	1.64	0.93	0.48
150x30	1.41	1.24	0.46	<b>0.26</b>	1.33	1.45	0.67
250x10	0.87	0.65	0.25	<b>0.15</b>	0.97	0.71	0.25
250x20	1.01	0.82	0.40	<b>0.23</b>	0.40	0.90	0.58
250x30	1.00	0.77	0.36	<b>0.18</b>	1.18	0.89	0.20
350x10	0.63	0.48	0.23	<b>0.12</b>	0.71	0.18	0.35
350x20	0.68	0.47	0.30	<b>0.15</b>	0.92	0.58	0.33
350x30	0.78	0.65	0.24	<b>0.10</b>	0.96	0.84	0.34
<i>ARPD</i>	1.09	0.93	0.36	<b>0.23</b>	0.95	0.87	0.50
<i>AvTime</i>	184.93	102.25	73.1	60.5	<b>19.0129</b>	19.5462	20.2563

As mentioned in sub-section 5.2, 5.3 and 5.4, the hybridization process works on exchanging roles and investing the ability of SBH algorithms (GD, TS and VNS). This process in question is conducted to exploit the search space for the purpose of enhancing the deficit in the genetic algorithm. On the other hand, the ability of a genetic algorithm for exploring the search space is taken advantage of to enhance the deficit in SBH algorithms.

## 7. CONCLUSION AND FUTURE WORK

This work proposed a hybridization of genetic algorithm with single-based metaheuristic algorithms. The proposed methods able to efficiently and effectively produce quality results. The reason behind the success

of our algorithms is because of the combination of the population-based metaheuristic represented by the genetic algorithm and the single-based metaheuristic represented by great deluge algorithm, tabu search algorithm and variable neighborhood search algorithm. The population-based metaheuristics, which are powerful in the exploration of the search space and weak in the exploitation of the solutions, will try to optimize solutions globally. The single-based metaheuristics, are powerful optimization methods in terms of exploitation, they will try to optimize solutions locally. Despite the quality of the results obtained, especially the GGA-VNS, it still needs to be improved to get better results or match the results of the literature. In the future, an adaptive hybrid GGA is suggested to select the suitable single-based metaheuristic algorithm automatically in order to obtain better results.

## ACKNOWLEDGEMENT

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




## REFERENCES

- [1] H. Y. Fuchigami and S. Rangel, "A survey of case studies in production scheduling: Analysis and perspectives," *Journal of Computational Science*, vol. 25, pp. 425–436, 2018, doi: 10.1016/j.jocs.2017.06.004.
- [2] F. Villa, E. Vallada, and L. Fanjul-Peyro, "Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource," *Expert Systems with Applications*, vol. 93, pp. 28–38, 2018, doi: 10.1016/j.eswa.2017.09.054.
- [3] C. Blum and A. Roli, "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003, doi: 10.1145/937503.937505.
- [4] R. J. O. and K. Hoffman, "Exact methods for solving traveling salesman problems with pickup and delivery in real time," *Optimization-Online*, vol. 9, 2018.
- [5] Y. Zhao, D. Jiao, and J. Mao, "Fast Nested Cross Approximation Algorithm for Solving Large-Scale Electromagnetic Problems," *IEEE Transactions on Microwave Theory and Techniques*, vol. 67, no. 8, pp. 3271–3283, 2019, doi: 10.1109/TMTT.2019.2920894.
- [6] V. J. Rayward-Smith, C. Osman, C. R. Reeves, and G. D. Smith, *Modern heuristic search methods*. 1996.
- [7] K. Chakhlevitch and P. Cowling, "Hyperheuristics: Recent developments," *Studies in Computational Intelligence*, vol. 136, pp. 3–29, 2008, doi: 10.1007/978-3-540-79438-7\_1.
- [8] J. Blazewicz, J. K. Lenstra, and A. H. G. R. Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983, doi: 10.1016/0166-218X(83)90012-4.
- [9] M. L. Pinedo, "Scheduling: Theory, algorithms, and systems: Fourth edition." *Scheduling: Theory, Algorithms, and Systems: Fourth Edition*, vol. 9781461423, pp. 1–673, 2012, doi: 10.1007/978-1-4614-2361-4.
- [10] H. Kellerer and V. A. Strusevich, "Scheduling problems for parallel dedicated machines under multiple resource constraints," *Discrete Applied Mathematics*, vol. 133, no. 1–3, pp. 45–68, 2003, doi: 10.1016/S0166-218X(03)00433-5.
- [11] A. Mendsiek, J. N. D. Gupta, and J. Herrmann, "Scheduling identical parallel machines with fixed delivery dates to minimize total tardiness," *European Journal of Operational Research*, vol. 243, no. 2, pp. 514–522, 2015, doi: 10.1016/j.ejor.2014.12.002.
- [12] W. C. Yeh, M. C. Chuang, and W. C. Lee, "Uniform parallel machine scheduling with resource consumption constraint," *Applied Mathematical Modelling*, vol. 39, no. 8, pp. 2131–2138, 2015, doi: 10.1016/j.apm.2014.10.012.
- [13] A. Grigoriev, M. Sviridenko, and M. Uetz, "Unrelated parallel machine scheduling with resource dependent processing times," *Lecture Notes in Computer Science*, vol. 3509, pp. 182–195, 2005, doi: 10.1007/11496915\_14.
- [14] Y. Guo, A. Lim, B. Rodrigues, and Y. Liang, "Minimizing the makespan for unrelated parallel machines," *International Journal on Artificial Intelligence Tools*, vol. 16, no. 3, pp. 309–415, 2007, doi: 10.1142/s0218213007003175.
- [15] L. Fanjul-Peyro and R. Ruiz, "Size-reduction heuristics for the unrelated parallel machines scheduling problem," *Computers and Operations Research*, vol. 38, no. 1, pp. 301–309, 2011, doi: 10.1016/j.cor.2010.05.005.
- [16] D. Yilmaz Eroglu, H. C. Ozmutlu, and S. Ozmutlu, "Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times," *International Journal of Production Research*, vol. 52, no. 19, pp. 5841–5856, 2014, doi: 10.1080/00207543.2014.920966.
- [17] L. Fanjul-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, 2017, doi: 10.1016/j.ejor.2017.01.002.
- [18] T. Arbaoui and F. Yalaoui, "Solving the Unrelated Parallel Machine Scheduling Problem with Additional Resources Using Constraint Programming," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10752 LNAI, pp. 716–725, 2018, doi: 10.1007/978-3-319-75420-8\_67.
- [19] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Approximation algorithms for scheduling on multiple machines," *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, vol. 2005, pp. 254–263, 2005, doi: 10.1109/SFCS.2005.21.
- [20] A. Grigoriev, M. Sviridenko, and M. Uetz, "LP rounding and an almost harmonic algorithm for scheduling with resource dependent processing times," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4110 LNCS, pp. 140–151, 2006, doi: 10.1007/11830924\_15.
- [21] E. B. Edis and C. Oguz, "Parallel machine scheduling with additional resources: A lagrangian-based constraint programming approach," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6697 LNCS, pp. 92–98, 2011, doi: 10.1007/978-3-642-21311-3\_10.
- [22] E. B. Edis and C. Oguz, "Parallel machine scheduling with flexible resources," *Computers and Industrial Engineering*, vol. 63, no. 2, pp. 433–447, 2012, doi: 10.1016/j.cie.2012.03.018.
- [23] L. Fanjul, F. Perea, and R. Ruiz, "Algorithms for the unspecified unrelated parallel machine scheduling problem with additional resources," *Proceedings of 2015 International Conference on Industrial Engineering and Systems Management, IEEE IESM 2015*, pp. 69–73, 2016, doi: 10.1109/IESM.2015.7380139.




- [24] 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, 2018, doi: 10.1016/j.ejor.2018.05.056.
- [25] E. Vallada, F. Villa, and L. Fanjul-Peyro, "Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem," *Computers and Operations Research*, vol. 111, pp. 415–424, 2019, doi: 10.1016/j.cor.2019.07.016.
- [26] J. Y. T. Leung, "Handbook of scheduling: Algorithms, models, and performance analysis," *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pp. 1–1195, 2004.
- [27] J. Błażewicz, N. Brauner, and G. Finke, "Scheduling with discrete resource constraints," *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pp. 18–23, 2004.
- [28] J. Błażewicz, k. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, "Scheduling under Resource Constraints," *Handbook on Scheduling*, pp. 425–475, 2007, doi: 10.1007/978-3-540-32220-7\_12.
- [29] 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, 2006, doi: 10.1016/j.ejor.2004.12.008.
- [30] 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, 2018, doi: 10.1007/s10845-015-1117-6.
- [31] 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, 2011, doi: 10.1080/03052151003759117.
- [32] M. Pinedo, "Scheduling: theory, and systems (3rd ed.)," *Springer*, 2008.
- [33] E. G. Talbi, "Metaheuristics: From Design to Implementation," *Metaheuristics: From Design to Implementation*, 2009, doi: 10.1002/9780470496916.
- [34] 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.
- [35] O. Dib, M. A. Manier, L. Moalic, and A. Caminada, "Combining VNS with Genetic Algorithm to solve the one-to-one routing issue in road networks," *Computers and Operations Research*, vol. 78, pp. 420–430, 2017, doi: 10.1016/j.cor.2015.11.010.
- [36] M. Forsberg, "Local search hybridization of a genetic algorithm for solving the University Course Timetabling Problem," 2018.
- [37] M. H. Abed and M. N. M. Kahar, "Guided genetic algorithm for solving unrelated parallel machine scheduling problem with additional resources," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 26, no. 2, p. 1036, May 2022, doi: 10.11591/ijeecs.v26.i2.pp1036-1049.
- [38] M. H. Hassan, S. Kamel, S. Q. Salih, T. Khurshaid, and M. Ebeed, "Developing chaotic artificial ecosystem-based optimization algorithm for combined economic emission dispatch," *IEEE Access*, vol. 9, pp. 51146–51165, 2021, doi: 10.1109/ACCESS.2021.3066914.
- [39] C. Y. Zhang, P. G. Li, Z. L. Guan, and Y. Q. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem," *Computers and Operations Research*, vol. 34, no. 11, pp. 3229–3242, 2007, doi: 10.1016/j.cor.2005.12.002.
- [40] G. Zäpfel, R. Braune, and M. Bögl, "Metaheuristic search concepts: A tutorial with applications to production and logistics," *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*, pp. 1–316, 2010, doi: 10.1007/978-3-642-11343-7.
- [41] G. Kendall and M. Mohamad, "Channel assignment in cellular communication using a great deluge hyper-heuristic," *Proceedings - IEEE International Conference on Networks, ICON*, vol. 2, pp. 769–773, 2004, doi: 10.1109/ICON.2004.1409283.
- [42] Y. J. Gong, J. Zhang, O. Liu, R. Z. Huang, H. S. H. Chung, and Y. H. Shi, "Optimizing the vehicle routing problem with time windows: A discrete particle swarm optimization approach," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, no. 2, pp. 254–267, 2012, doi: 10.1109/TSMCC.2011.2148712.
- [43] R. R. García, "Informative talk city of arts and sciences (in Catalan: Charla divulgativa ciutat de les arts y les ciencies)," *Sistemas de optimización aplicada*, 2019. <http://soa.iti.es/>.

## 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 Nasional, 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](mailto:munt1979@yahoo.com).



**Mohd Nizam Mohamad 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](mailto:mnizam@ump.edu.my).