# Heuristic Embedded Genetic Algorithm for Heterogeneous Project Scheduling Problems

Firoz Mahmud[a], Forhad Zaman[b], Ruhul Sarker[b], Daryl Essam[b]
*School of Engineering and Information Technology*
*University of New South Wales*
Canberra, Australia
firoz.mahmud@student.adfa.edu.au[a], {f.zaman, r.sarker, d.essam}@unsw.edu.au[b]

***Abstract*—Over the last few decades, many solution approaches have been developed for solving different variants of resource-constrained project scheduling problems (RCPSPs). In most of them, it is assumed that a project consists of some homogeneous activities that require all types of resources over the entire project horizon. On the contrary, many real-world projects consist of heterogeneous activities that use different types of resources at different time instants during the project execution. The application of existing approaches, developed for RCPSPs with homogeneous activities, in solving RCPSPs with heterogeneous activities is computationally expensive. In this paper, we propose a heuristic embedded genetic algorithm to address RCPSPs with heterogeneous activities. Two heuristics are proposed to obtain high-quality feasible solutions. The first heuristic is based on priority rules while the second one based on a new neighbourhood swapping matrix. To evaluate the performance of the proposed algorithm, we solve a number of real-world and modified test problems, and the obtained results are compared with an existing algorithm. It is found that the proposed approach obtains high-quality solutions with a significantly lower computational time compared to other algorithms.**

***Index Terms*—heterogeneous project scheduling problems, evolutionary algorithms, multi-operator algorithms**

## I. Introduction

During the last few decades, project scheduling problems have been widely studied due to their importance in the various real-world situations, such as aircraft scheduling, job shop, manufacturing, and constructions projects [1]. The primary objective in such scheduling problem is to complete the tasks with minimum possible time and resources. Every project consists of a number of activities, with each activity must be completed during the project implementation while satisfying their temporal relationship, which means an activity cannot start until its all predecessor are finished. Besides, each activity requires one or more resources for a certain duration, while the maximum availability of each resource is limited and given. Therefore, the resources must be optimally allocated so that the best outcome can be obtained. This variant of the project scheduling problem is known as the Resource-Constrained Project Scheduling Problem (RCPSP), in which the objective is to minimise its makespan by determining the best schedule and satisfying all precedence and resource constraints.

As RCPSP is an NP-hard optimisation problem [2], a large number of solution approaches have been introduced for solving different variants of RCPSP. The solution approach can be broadly categorised into four types: i) exact algorithms, ii) heuristic algorithms, iii) meta-heuristic algorithms, and (iv) hybrid meta-heuristic based algorithms. For the exact approaches, integer programming [3], mixed-integer linear programming (MILP) [4], constraint programming [5], branch-and-bound (BB) [6] are the popular approaches. However, for the large scale problems, these approaches are not only computationally expensive but also unable to achieve optimal or near-optimal solutions [7], [8].

For solving large-scale problems, heuristic and meta-heuristic algorithms, such as simulated annealing (SA) [9], [10], tabu search [11], [12], genetic algorithm (GA) [13], [14], differential evolution (DE) [15], scatter search (SS) [16], [17], artificial immune system (AIS) [18], [19], ant colony optimization (ACO) [20], [21], and particle swarm optimization (PSO) [22], have gained popularity over the last few years. However, many heuristic-based solutions approaches often trap in a local optima. To avoid that, some meta-heuristic approaches use an iterative technique, namely a random perturbation. Alcaraz et al. [13] proposed a robust GA based on the activity list, where two serial generation schemes (SGSs) are used, namely forward and backward SGSs. They used an additional gene to decide whether a forward or backward SGS is applied in generating a schedule from the activity list. They have also included a local search approach with GA. The quality of their solutions for large-scale problems are not superior.

As no single algorithm shows better performance for a wide-rage of RCPSPs, some researchers used multiple meta-heuristic approaches which is called hybrid meta-heuristic. It is often defined as hybrid meta-heuristic as a skilled combination of a meta-heuristic with another optimisation technique. Elsayed et al. [1] developed a new hybrid meta-heuristic algorithm named as consolidated optimisation algorithm (COA) for solving single-mode RCPSP, while Zaman et al. [23] extended that COA to solve multi-mode RCPSP. Multi-mode RCPSP (MM-RCPSP) is an extension of RCPSPs, where each activity can execute under several available modes, each mode has a different duration and resource requirement to execute an activity. Both papers used two different multi-operator evolutionary algorithms, namely, multi-operator genetic algorithm (MOGA) and multi-operator differential evolution (MODE), in which each uses their own sub-population and evolve own individuals to generate new offspring. In both approaches, a local

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

search approach is used to convert new infeasible offspring to feasible ones. The performances of both approaches have been demonstrated by solving different standard and real-world test problems. Bettemir et al. [24] proposed a hybrid meta-heuristic optimisation framework using two algorithms, namely GA and SA. They integrated GA for parallel search capability with the fine-tuning ability of SA to achieve a better algorithm for the RCPSP. They tested their proposed method using benchmark test problems and claimed the best solution. Fang et al. [25] proposed a new meta-heuristic framework, based on two different algorithms (i.e., memetic GA and PSO) for large-scale RCPSPs. In that framework, the virtual frog is encoded using an extended activity list (EAL) with a feasible solution and decoded it by the shuffled frog-leaping algorithm (SFLA) serial schedule generation scheme (SSGS). The algorithm was tested against the PSPLIB test problems and obtained better results for j60 and j120 benchmark sets. Tseng et al. [26] also proposed a hybrid meta-heuristic approach, where three different algorithms are considered, such as ACO, GA, and local search strategy. ACO algorithm searches the solution space and produces an activity list. Then, the list is used as the initial population for GA, with the GA continues to execute until the best solution is found. In addition, in that approach, a local search is used to improve the solution obtained from ACO and GA. They solved a wide range of RCPSPs standard benchmark sets, where the proposed one obtained satisfactory results compared to many other algorithms.

Most of the above-mentioned algorithms are designed for the standard RCPSPs benchmark problems where all activities are of homogeneous type that requires all types of resources. That means, there is a strict assumption that each and every activity requires all of the resources. However, in real-world problems, most activities are of heterogeneous type in which one activity may require one type of resource while another activity may require a different type of resource. For example, in a building construction project, foundation of the building will require a certain type of resources and electrical appliances require a different type of resources. Existing algorithms can be applied to solve such heterogeneous problems but they will be computationally expensive due to dealing with redundant resource constraints. To the best of authors' knowledge, no research is done to solve such RCPSPs of consist of heterogeneous activities.

In this paper, we consider such variant of real-world RCPSPs and propose an efficient solution approach to solve them. In the solution process, firstly, an initial schedule is randomly generated. Then a heuristic based on the earliest start time (EST) is used to obtain a feasible schedule if any infeasible solution exists. Secondly, the quality of a feasible schedule is improved by using a new neighbourhood swapping technique. Finally, a MOGA is applied to evolve the individuals. The proposed algorithm is verified by solving a set of real-world and modified test problems. For the comparison purpose, we also solve the problems using a well-known COA [1]. The obtained results are compared with those from our proposed approach, in which the proposed one outperforms COA.

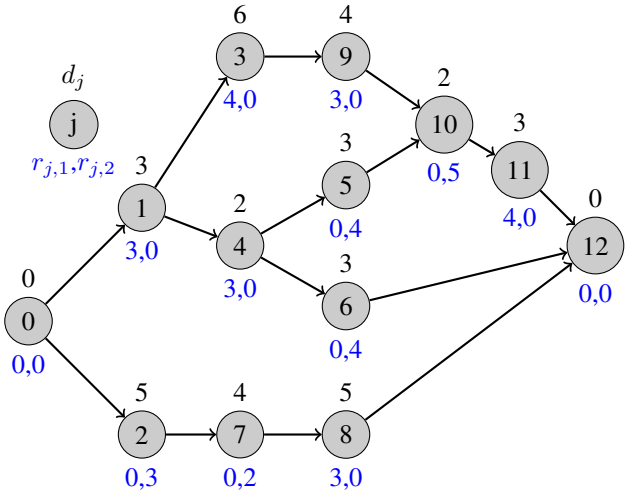


Fig. 1: An example network for the RCPSP.

The rest of this research paper is organised as follows: Section II describes the problem description, Section III represents mathematical model, Section IV discusses propose approach, Section V shows the experimental setup and result analysis, and Section VI represent conclusion and further research direction.

## II. Problem Description

The RCPSP can be formulated as follows: a project $S$ consists of $M+1$ activities, where all activities have to execute in order to complete the project. The project is defined as, $S = 0, 1, ..., M+1$, and resource set $R$ with $K$ renewable resource as $R = 1, 2, ..., K$. Each activity $M$ has a duration represented as $d_j$, where $j = 0, 1, ..., M+1$. Two dummy activities: *project start* and *project finish* are represented as $0$ and $M+1$, respectively. Each activity is scheduled subject to satisfying two constraints. Firstly, precedence constraint, which ensures that an activity cannot be started before finishing its predecessors. The second one is for the resource constraint, which means at a particular time period, total resource usages by the activities must not exceed their maximum availability. An example of a RCPSP, with modified resource requirement, is shown in Fig. 1, where *node 1* and *node 13* are dummy activity. $r_{j,1}$ and $r_{j,2}$ represent first and second resource demands of $j^{th}$ activity, respectively, with their maximum resource availability is $5$ for both resources. A feasible schedule, with makespan $C_{max} = 29$, is shown in Fig. 2. However, the schedule is shown in Fig. 2 can be improved, with rescheduling the activities which is shown in Fig. 3, where the makespan is $C_{max} = 23$.

## III. Mathematical Model

In this section, the optimisation problem with the objective is to minimise project makespan is discussed. In this model, we consider the following assumptions: 1) all activities of a project must be executed; 2) the activity duration is given in advance; 3) preemption is not allowed, which means an

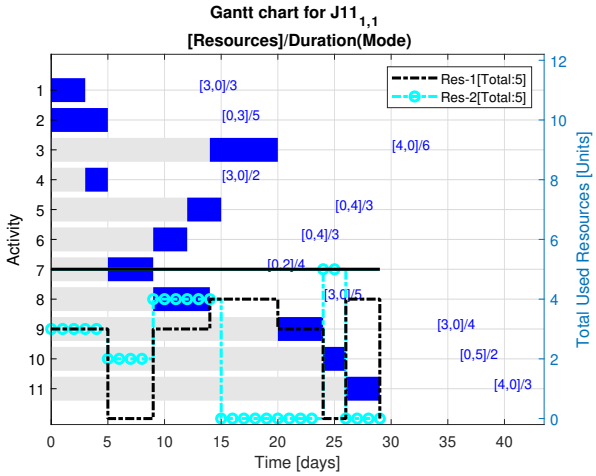


Fig. 2: A feasible schedule of Fig. 1 example network

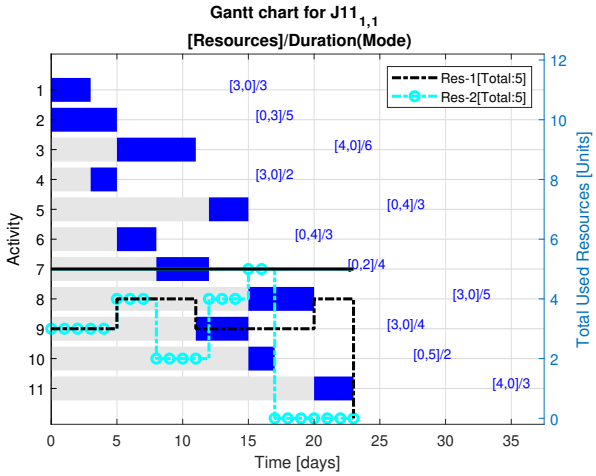


Fig. 3: Best schedule of Fig. 1 example network

activity cannot be interrupted during execution; 4) an activity cannot start until all predecessors have completely been finished; 5) at a certain time, an activity can use only one resource; and 6) the objective of this work is to minimise the makespan or total project duration. The objective of this optimisation problem can be expressed as:

$$\text{Min: } FT_{M+1} \tag{1}$$

Subject to:

$$FT_k \leq FT_j - d_j, \forall (j,k) \in \{0,1,...,M+1\} \tag{2}$$

$$\sum_{j \in A_t} r_{kj} \leq R_k, \ \forall k = 1,2,...,K \tag{3}$$

$$FT_j \geq 0, \ \forall j = 0,1,...,M+1 \tag{4}$$

where $FT_{M+1}$ is the finish time (FT) of the last dummy activity, $r_{kj}$ is used to represent the required $k^{th}$ type of resource of the $j^{th}$ activity, which must be less than or equal to the maximum available $k^{th}$ type of resource, $K$ is the number of resource types, and $A_t$ is a set of activities scheduled at time, t $\leq$ $FT_{M+1}$. Eqn. (1) represents the objective function, and the goal is to minimise FT of the last dummy activity. Eqn. (2) is for precedence relationship between the activities, that is, an activity cannot start before its predecessors. Eqn. (3) represents the resources constraint, and Eqn. (4) represents the finish time of any activity must be greater than or equal to zero.

## IV. Proposed Approach

In this research, we propose an evolutionary framework based on a MOGA and two heuristics, to solve a new variant of RCPSP, where the activities are heterogeneous type. It means, all activities of a project do not require all types of available resources, while each activity uses one type of resource. We name the proposed framework as 'H-MOGA', with its steps are described in Algorithm 1.

H-MOGA starts with randomly generated initial population of size $N_P$, where each population represents a random schedule of activities, as shown in subsection IV-A. The random schedule may be infeasible in terms of precedence and resources constraints. Any infeasible schedule is rectified to a feasible one using a heuristic, discussed in Algorithm 2. It is developed based on earliest start time (EST) and critical path method, in which an activity is scheduled as earliest as possible, subject to precedence and resource constraints. The obtained feasible schedule is further improved using the second heuristic, which is developed based on a swap eligible matrix, as discussed in Algorithm 3. Once both heuristics are applied, the makespan of each solution is evaluated, using Eqn. (1). If the best makespan is not a known optimal solution, a new set of schedules is generated based on MOGA operators, as discussed in IV-D. The newly generated schedules can be infeasible, and convert them into feasible by using Algorithm 2 and Algorithm 3, as discussed above. The process continues until the best solution is improved. The pseudo-code of H-MOGA is given in Algorithm 1, and its components are described in following subsections.

### A. Representation and Initial Generation

In this research, we represent the decision variable, $\overrightarrow{X}_i$, is the sequence of the activities, with the size of $\overrightarrow{X}_i$ is $M+1$. The initial population is randomly generated as follow:

$$\overrightarrow{X}_i = \{X_0, X_j, ..., X_{M+1}\}, i = 0,1,...,N_P - 1 \tag{5}$$

where $X_j$ is the $j^{th}$ non-dummy activity, while $X_0$ and $X_{M+1}$ are two dummy activities. $\overrightarrow{X}_i$ is the $i^{th}$ schedule, which are the random combinations of non-dummy activities.

### B. Heuristic 1: EST based Scheduling

Once a $\overrightarrow{X}_i$ is randomly generated that can be infeasible in terms of precedence and resources constraints. To convert any infeasible $\overrightarrow{X}_i$ to a feasible $\overrightarrow{X}_i$, heuristic-1 is used. In it, firstly, $EST_j$ and $FT_j$ (finish time of the $j^{th}$ activity) of the $j^{th}$ activity are calculated using Eqns. (6) and (7), respectively, without considering any resource constraints. Then, the activities which use same type of resource, are

**Algorithm 1** Pseudo-code of H-MOGA

**Require:** Number of runs (maxRuns), current fitness function ($cfe$), maximum fitness evaluation ($cfe_{max}$), $N_P$.

Compute swap eligible matrix, as discussed in Algorithm 3
**for** q=1 : maxRuns **do**
  $cfe = 1$
  Randomly generate initial schedule, $\overrightarrow{X}_i, i = 0, 1, ..., N_P - 1$, as discussed in subsection IV-A.
  If any $\overrightarrow{X}_i, i \in N_P$ is infeasible, obtain feasible using Algorithm 2. Update, $cfe \leftarrow cfe + N_P$ .
  Evaluate makespan of each $\overrightarrow{X}_i$ using Eqn. (1) .
  Improve $\overrightarrow{X}_i, i \in N_P$ using Algorithm 3. Update, $cfe \leftarrow cfe + N_P$ .
  **if** $cfe < cfe_{max}$ **then**
    Generate new $N_P$ offspring using the MOGA operators as discussed in subsection IV-D.
    Evaluate makespan of the offspring based on steps 5 and 6.
  **end if**
  Determine the best schedule based on the minimum makespan.
**end for**

verified with the resource availability. If any activity violates resource constraints, it reschedules at the next earliest possible time period. The details steps of this heuristic is shown in Algorithm 2.

$$EST_j = \begin{cases} 0, & \text{if } j \text{ has no predecessor} \\ max\big(FT_n | n \in P(j)\big), & 0 < j \leq M+1 \end{cases} \tag{6}$$

$$FT_j = EST_j + d_j \tag{7}$$

where $j$ is the $j^{th}$ activity, $FT_n$ is the finish time of $n^{th}$ predecessor from a set of predecessor of $j^{th}$ activity, and $M+1$ represents total number of activity. Eqn. (7) is used to find out the finish time of each activity, where $d_j$ represents duration of the $j^{th}$ activity.

### *C. Heuristic 2: Neighbourhood Swapping based Scheduling*

In this subsection, the proposed neighbourhood swapping based heuristic is discussed, which is used to improve the quality of a feasible $\overrightarrow{X_i}$ that obtained from the heuristic-1 as discussed in subsection IV-B. In it, an improved $\overrightarrow{X_i}$ is obtained by swapping some eligible activities. To do, firstly, we create a swap matrix (SM) based on Eqn. (8), which is used to represent eligibility for swapping between two activities. When the value of a cell is '1', it means the pair activities are eligible to swap, otherwise they are ineligible. $SM$ is created based on the type of resource demand and precedence relationships

**Algorithm 2** Pseudo-code of EST based Scheduling

**Require:** An infeasible $\overrightarrow{X}_i, i \in N_P$, $M+1$, $N_P$.

$\overrightarrow{Y}_i \leftarrow []$, where $\overrightarrow{Y}_i$ is the feasible schedule of $\overrightarrow{X}_i$ in terms of precedence constraints.
**for** j = $\overrightarrow{X}_i$ **do**
  **if** $P_j \in \overrightarrow{Y}_i$, where $P_j$ is the precedence of the $j^{th}$ activity. **then**
    break.
  **end if**
  Calculate $EST_j$ and $FT_j$ using Eqn. (6) and (7).
  $\overrightarrow{Y}_i \leftarrow j$ and remove $j$ from $\overrightarrow{X_i}$
  Repeat steps 2 to 7 until all activities of $\overrightarrow{X_i}$ is scheduled.
**end for**
$\overrightarrow{Z}_i \leftarrow []$, where $\overrightarrow{Z}_i$ is the feasible schedule of $\overrightarrow{Y}_i$ in terms of resource constraints, and $t \leftarrow 0$, where t is the project start time
**for** j = 1 : noAct-1 **do**
  **if** $r_{kj} < R_t$ **then**
    $\overrightarrow{Z}_i \leftarrow j$, update, $R_t \leftarrow R_t - r_{kj}$ and $FT_j$.
  **else**
    Update $t \leftarrow t + 1$ until $j^{th}$ is successfully scheduled based on step 13.
    break.
  **end if**
**end for**
Evaluate makespan as finish time of the last dummy activity.

of the activities, with following four rules:

$$SEM_{k,j} = \begin{cases} 0, & \text{if } k == j \text{ and } R_k \neq R_j \\ 0, & \text{if } k > j,\ j \in P_k \text{ and } R_k \neq R_j \\ 0, & \text{if } j > k,\ k \in P_j \text{ and } R_k \neq R_j \\ 1, & \text{otherwise} \end{cases} \tag{8}$$

where $R_j$ and $R_k$ are the resource demands of the $j^{th}$ and $k^{th}$ activity, respectively, and $P_j$ represents the predecessors of the $j^{th}$ activity.

An example of a SM is shown in Eqn. (8) for the j11 test problem, as shown in Fig. 1. The value of all cells of the first row and first column are always '0', as first dummy activity, i.e. node 1 is the common predecessor for all other activities. Similarly, all cells of the last row and column are also '0', as node 13 is the last dummy activity. The value of a cell of the matrix is '1', where both activities have no direct or indirect precedence relationships and their resource demands are the same type. For example, SM(2,9) or SM(9,2) are '1', due to the activity 2 is not the direct or indirect predecessor of activity 9. Also, both activities 2 and 9 use same type of resource, which is the resource-1. On the other hand, SM(2,5) or SM(2,10) are '0', as activity 2 is the immediate predecessor of activity 5, and indirect predecessor of activity 10, though their resource demands are same type.

Once the SM is created of a project, the heuristic-2 is applied to improve the quality of a schedule. In it, a feasible $\overrightarrow{X}_i$ is modified by swapping two eligible activities in sequential order. Once a swap is performed, the new makespan is calculated. If the new makespan is better than the original one, the original schedule is replaced to the modified $\overrightarrow{X}_i$. Otherwise, the alternative $\overrightarrow{X}_i$ is discarded, and the original one is kept. The key steps of the heuristic-2 are shown in Algorithm 3.

$$SM = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (9)$$

**Algorithm 3** Neighbourhood Swapping based Scheduling

**Require:** A feasible $\overrightarrow{X}_i$, and its makespan, Number of Trails (numTrail).

1: Determine $SM$ using Eqn. (8).
2: **for** s=1 : numTrail **do**
3:  **if** numTrail>1 **then**
4:  Generate new $\overrightarrow{X}_i$, by swapping between two activities where value of a cell of $SM$ is '1'.
5:  Determine $makespan_{new}$ of the new $\overrightarrow{X}_i$, using Algorithm 2.
6:  **end if**
7:  **if** $makespan_{new} < makespan$ **then**
8:  Update $\overrightarrow{X}_i$.
9:  **end if**
10: **end for**
11: Determine the best $makespan$

### D. MOGA Operators

In this subsection, we discuss the operators of MOGA, that are used to generate offspring of $\overrightarrow{X}_i, i \in N_P$. We use two crossovers, namely uniform and two-point crossovers, and one left shift mutation operator. To create new offspring, firstly one of two crossover operator is selected based on the probabilities which are initially set to 0.5, with their probabilities are self-adaptively updated based on their performances to generate better individuals than their parents. Alternatively, the operator which performs better, its probability is increased, so a higher number of individuals are generated using that operator. In the two-point crossover, two-point is randomly selected for two random parents and then exchanged between their first and third portions with each other to produce two unique offspring. In the uniform crossover, a random number between 0 and 1, is generated, then the first offspring is generated from the first parent if the number is less than 0.5, while another offspring is generated from the other parent. For producing a more diverse population, a left shift mutation operator is used, in which a random activity is shifted to its left side. The details of all operators of MOGA can be found in [1].

## V. Experimental Results and Analysis

In this section, the performance evaluation of our proposed algorithm has been carried out for solving a wide range of standard and real-world test problems. The standard test problems are considered up to 100 activities, and real-world [27] problems up to 22 activities. For the comparison purpose, all test problems are also solved using the three variants of the proposed H-MOGA, and a state-of-the-art algorithm (i.e., COA), as:

- **var1:** In this variant, a schedule is randomly generated, and then obtained feasible ones based on EST while satisfying the precedence and resource constraints. For **var1**, we use steps 2 to 5 in Algorithm 1.
- **var2:** In this variant, the quality of the solution is improved using the SM as discussed in Algorithm 3. For **var2**, we use steps 1 to 6 in Algorithm 1.
- **var3:** In this variant, the quality of the solution is further improved using the MOGA, as described in subsection IV-D. For **var3**, we use all steps in Algorithm 1.
- **COA:** In this approach, a newly developed COA is employed to solve the new variant of RCPSPs. It is also worth to mention that COA outperformed for many existing algorithms for solving the well-known PSPLIB benchmark sets for RCPSPs. The details of COA and its parameters can be found in [1].

For a fair comparison, we run 31 times of each algorithm for all test problems, and their statistical results are reported. The population size and the maximum number of generations of all cases are set to 10 and 500 respectively, along with other parameters based on COA [1]. All algorithms have been implemented in Matlab2018b environment on a computer with a Core(TM)i7-8700 CPU@3.20GHz, 16GM RAM, with Windows 10 operating system.

### A. Test Problems

In this subsection, we solve different standard test problems, which is originally taken from [3], [27]. As in a standard test problem, each activity requires all types of resources; we modify the original test problems to satisfy the requirements so that each activity requires one resource rather than all of them. The resource availability of all types of resources is set to 5 for all test problems. The new resource requirement of the considered test problems can be found in Appendix A.

Each test problem is solved using the above four algorithms 31 times, and their minimum (Min), Median, Mean, maximum (Max), standard deviation (STD), average standard deviation

TABLE I: Comparison of the performances of different algorithms for j11 test problem

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 23.00 | 23.00 | 23.00 | 23.00 | 0.00 | 27.78 | 125.11 |
| Var1 | 23.00 | 26.00 | 26.32 | 31.00 | 2.56 | 27.78 | 0.01 |
| Var2 | 23.00 | 25.00 | 24.93 | 29.00 | 2.31 | 27.78 | 0.03 |
| Var3 | 23.00 | 23.00 | 23.00 | 23.00 | 0.00 | 27.78 | 25.73 |

TABLE II: Comparison of the performances of different algorithms for j20 test problem

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 39.00 | 39.00 | 39.00 | 39.00 | 0.00 | 39.28 | 268.18 |
| Var1 | 39.00 | 40.00 | 41.42 | 46.00 | 2.16 | 39.28 | 0.02 |
| Var2 | 39.00 | 40.00 | 40.74 | 45.00 | 1.48 | 39.28 | 0.08 |
| Var3 | 39.00 | 39.00 | 39.00 | 39.00 | 0.00 | 39.28 | 33.31 |

(%AvTD) of the makespan, and the computational time is reported in Tables I, II, III, IV, V, and VI for j11, j20, j30, j50, j75, and j100, respectively. The % AvTD is obtained using the following equation:

$$\%AvTD = 100 \times \frac{Min_{Makespan} - OPT_{LB}}{OPT_{LB}} \qquad (10)$$

where $Min_{Makespan}$ is the minimum makespan of a particular schedule and $OPT_{LB}$ is the critical path optimum value.

From Tables I to VI, it is seen that mean values of var1 and var2, are higher than that of var3 and COA for all cases, with var3 and COA are almost similar. However, regarding the computational time, it is seen that COA is the most expensive approach, with H-MOGA (var3) obtains similar quality solution with a significantly less computational time. If a system is very sensitive about the computational time, var1 is the best approach as it obtains feasible solutions with the least computational time. However, it quality cannot be guaranteed. Var2 is a modest approach as it takes more time than that of var1, which obtains better average results than that of var1. On the other hand, var3, which is our proposed approach that provides the best results for all cases, with a reasonable computational time. It is worth to mention that, var3 obtains the same quality solution as COA obtains, while var3 is up to eight times faster than COA.

TABLE III: Comparison of the performances of different algorithms for j30 test problem

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 56.00 | 56.00 | 56.00 | 56.00 | 0.00 | 180.00 | 727.54 |
| Var1 | 57.00 | 65.00 | 65.64 | 76.00 | 3.90 | 185.00 | 0.06 |
| Var2 | 56.00 | 62.00 | 63.35 | 72.00 | 3.66 | 180.00 | 0.45 |
| Var3 | 56.00 | 56.00 | 56.25 | 58.00 | 0.51 | 180.00 | 177.78 |

TABLE IV: Comparison of the performances of different algorithms for j50 test problem

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 108.00 | 108.00 | 108.00 | 108.00 | 0.00 | 145.45 | 1041.40 |
| Var1 | 108.00 | 129.00 | 129.00 | 150.00 | 9.16 | 145.45 | 0.14 |
| Var2 | 108.00 | 119.00 | 121.13 | 137.00 | 7.55 | 145.45 | 0.74 |
| Var3 | 108.00 | 108.00 | 108.00 | 110.00 | 0.40 | 145.45 | 193.82 |

TABLE V: Comparison of the performances of different algorithms for j75 test problem

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 150.00 | 150.00 | 150.00 | 150.00 | 0.00 | 341.18 | 2520.70 |
| Var1 | 151.00 | 163.00 | 166.00 | 186.00 | 8.61 | 344.12 | 0.23 |
| Var2 | 151.00 | 161.00 | 162.67 | 176.00 | 6.36 | 344.12 | 0.29 |
| Var3 | 150.00 | 153.00 | 151.55 | 153.00 | 1.52 | 341.18 | 545.94 |

TABLE VI: Comparison of the performances of different algorithms for j100 test problem

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 192.00 | 192.00 | 192.00 | 192.00 | 0.00 | 380.00 | 4103.20 |
| Var1 | 192.00 | 207.00 | 206.55 | 226.00 | 7.94 | 380.00 | 0.40 |
| Var2 | 192.00 | 203.00 | 203.32 | 218.00 | 6.18 | 380.00 | 0.59 |
| Var3 | 192.00 | 192.00 | 192.00 | 192.00 | 0.00 | 380.00 | 829.35 |

### *B. Real-World Test Problems*

To show the suitability of the proposed algorithm, in this subsection, we consider two real-world RCPSPs; namely, i) family day test problem with 22 activities and ii) railway station Sint-Joost with 18 activities. The data of both problems can be found in [27]. Both problems are solved using the four algorithms for 31 times, and their results are reported in Tables VII and VIII. Although both COA and var3 obtain the same results for j22, var3 outperforms than COA in terms of computational time. For j18 problem, var3 obtains better results than that of other variants and COA. The average standard deviation shows the superiority of our proposed variant, i.e., var3. Moreover, H-MOGA (var3) is significantly faster than that of COA, with at least three and a half times faster than COA.

### *C. Statistical Test*

In this subsection, we use two popular statistical tests: Wilcoxon and Friedman sign tests, to validate the performance of the proposed algorithm. Firstly, Wilcoxon test is carried out based on the average values of the makespan for all test problems, with the result is shown in Table IX. In it, $p$

TABLE VII: Comparison of the performances of different algorithms for j22 test problem [Real-World Data]

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 191.00 | 191.00 | 191.00 | 191.00 | 0.00 | 148.05 | 999.70 |
| Var1 | 191.00 | 191.00 | 191.29 | 192.00 | 0.46 | 148.05 | 0.07 |
| Var2 | 191.00 | 192.00 | 191.51 | 192.00 | 0.51 | 148.05 | 0.48 |
| Var3 | 191.00 | 191.00 | 191.00 | 191.00 | 0.00 | 148.05 | 219.93 |

TABLE VIII: Comparison of the performances of different algorithms for j18 test problem [Real-World Data]

| Algorithm | Min | Median | Mean | Max | STD | %AvTD | Time (sec) |
|---|---|---|---|---|---|---|---|
| COA | 133.00 | 133.00 | 133.00 | 133.00 | 0.00 | 6.40 | 291.73 |
| Var1 | 133.00 | 141.00 | 142.13 | 163.00 | 6.92 | 8.00 | 0.02 |
| Var2 | 127.00 | 135.00 | 135.13 | 143.00 | 4.14 | 1.60 | 0.08 |
| Var3 | 127.00 | 127.00 | 127.39 | 132.00 | 1.23 | 1.60 | 80.55 |

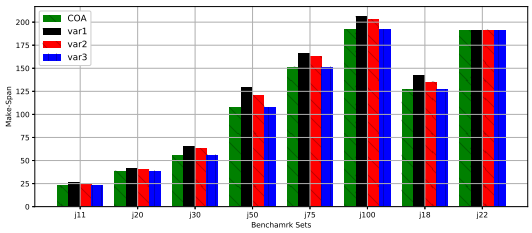


Fig. 4: Makespan comparison between different algorithms

TABLE IX: Wilcoxon sign test results for different algorithms

| Algorithms | Criterion | p | Decision |
|---|---|---|---|
| Var3 vs Var1 | Mean Makespan | 0.012 | + |
| Var3 vs Var2 | Mean Makespan | 0.012 | + |
| Var3 vs COA | Mean Makespan | 0.317 | - |

value is used to represent the significance of two data sets, with $p < 0.05$ (5% significance) indicates that there is a significant difference between two data sets. We conclude a decision from $p$ value by using three signs: '+', '-', and '$\approx$', where '+' sign represents that the performance of the first algorithm is significantly better than the second algorithm, '-' sign represents worse, and '$\approx$' sign represents no significant difference. According to Table IX, var3 is better than that of var1 and var2, though very competitive with COA. However, in term of time taken to solve a problem, var3 is much better than that of COA.

In addition, we use the non-parametric Friedman test to rank different algorithms based on the best values of the makespan for all test problems. The mean ranks are shown in Table X, with a higher value, means the algorithm is better than others. From Table X, var3 has the highest rank's value, which represents the superiority of our proposed algorithm.

## VI. Conclusion and Future Direction

In this research, we proposed H-MOGA based on a MOGA and two heuristics, for solving practical RCPSPs that consists of heterogeneous activities. In tradition, RCPSP consists of many homogenous activities, which require similar types of resources over the project horizon. In our case, all activities must not use all types of available resources of a project. Instead, one activity requires one type of resource, while others use different types of resources. In the solution approach, we propose two different heuristics to obtain high-quality solution quickly. The first heuristic based on EST and critical path method are used to rectify an infeasible schedule. The second heuristic based on a swapping matrix is used to improve the quality of the feasible schedules. In addition, a MOGA is used to obtain the best solutions by avoiding local optima.

TABLE X: Friedman test ranks for different algorithms

| Criteria | Var1 | Var2 | Var3 | COA |
|---|---|---|---|---|
| Mean Makespan | 2.31 | 2.31 | **2.81** | 2.56 |

For the experimental study, we consider both real-world and modified test problems, in which each activity requires a single type of resource rather than all. Also, for the comparison purpose, we solve all test problems using the three variants of H-MOGA and a state-of-the-art algorithm. It is found that the proposed approach obtains high-quality solutions within a reasonable computational time. It is worth to mention that H-MOGA obtains the best quality solution for a real-world (i.e., j18) problem, with spending three times less computational time that of the standard COA algorithm. The makespan comparison of our proposed variants with an existing algorithm is shown in Fig. 4.

In future, this research can be extended by considering uncertainty in the test problems.

## VII. Acknowledgement

The research is partly supported by a Australian Research Council project ARC DP190102637.

## References

[1] S. Elsayed, R. Sarker, T. Ray, and C. C. Coello, "Consolidated optimization algorithm for resource-constrained project scheduling problems," *Information Sciences*, vol. 418, pp. 346–362, 2017.

[2] M. R. Garey and D. S. Johnson, *Computers and intractability*, vol. 29. wh freeman New York, 2002.

[3] R. Kolisch and A. Sprecher, "Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program," *European journal of operational research*, vol. 96, no. 1, pp. 205–216, 1997.

[4] O. Koné, C. Artigues, P. Lopez, and M. Mongeau, "Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources," *Flexible Services and Manufacturing Journal*, vol. 25, no. 1-2, pp. 25–47, 2013.

[5] P. Laborie, "Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results," *Artificial Intelligence*, vol. 143, no. 2, pp. 151–188, 2003.

[6] N. Christofides, R. Alvarez-Valdés, and J. M. Tamarit, "Project scheduling with resource constraints: A branch and bound approach," *European Journal of Operational Research*, vol. 29, no. 3, pp. 262–273, 1987.

[7] L. Özdamar and G. Ulusoy, "A survey on the resource-constrained project scheduling problem," *IIE transactions*, vol. 27, no. 5, pp. 574–586, 1995.

[8] W. Herroelen and R. Leus, "Project scheduling under uncertainty: Survey and research potentials," *European journal of operational research*, vol. 165, no. 2, pp. 289–306, 2005.

[9] F. F. Boctor, "Resource-constrained project scheduling by simulated annealing," *International Journal of Production Research*, vol. 34, no. 8, pp. 2335–2351, 1996.

[10] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268–281, 2003.

[11] M. Verhoeven, "Tabu search for resource-constrained scheduling," *European Journal of Operational Research*, vol. 106, no. 2-3, pp. 266–276, 1998.

[12] L. Bukata, P. Šcha, and Z. Hanzálek, "Solving the resource constrained project scheduling problem using the parallel tabu search designed for the cuda platform," *Journal of Parallel and Distributed Computing*, vol. 77, pp. 58–68, 2015.

[13] J. Alcaraz and C. Maroto, "A robust genetic algorithm for resource allocation in project scheduling," *Annals of operations Research*, vol. 102, no. 1-4, pp. 83–109, 2001.

[14] M. Sebt, M. Afshar, and Y. Alipouri, "Hybridization of genetic algorithm and fully informed particle swarm for solving the multi-mode resource-constrained project scheduling problem," *Engineering Optimization*, vol. 49, no. 3, pp. 513–530, 2017.

[15] M.-Y. Cheng, D.-H. Tran, and Y.-W. Wu, "Using a fuzzy clustering chaotic-based differential evolution with serial method to solve resource-constrained project scheduling problems," *Automation in Construction*, vol. 37, pp. 88–97, 2014.
[16] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision sciences*, vol. 8, no. 1, pp. 156–166, 1977.
[17] H. Meng, B. Wang, Y. Nie, X. Xia, and X. Zhang, "A scatter search hybrid algorithm for resource availability cost problem," in *Harmony Search Algorithm*, pp. 39–51, Springer, 2016.
[18] V. Van Peteghem and M. Vanhoucke, "An artificial immune system algorithm for the resource availability cost problem," *Flexible services and manufacturing journal*, vol. 25, no. 1-2, pp. 122–144, 2013.
[19] P. Jedrzejowicz and E. Ratajczak-Ropel, "Experimental evaluation of a-teams solving resource availability cost problem," in *Intelligent Decision Technologies 2019*, pp. 213–223, Springer, 2020.
[20] R. Saremi, Y. Yang, and A. Khanfor, "Ant colony optimization to reduce schedule acceleration in crowdsourcing software development," in *International Conference on Human-Computer Interaction*, pp. 286–300, Springer, 2019.
[21] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, vol. 2, pp. 1470–1477, IEEE, 1999.
[22] H. Zhang, H. Li, and C. Tam, "Particle swarm optimization for resource-constrained project scheduling," *International Journal of Project Management*, vol. 24, no. 1, pp. 83–92, 2006.
[23] F. Zaman, S. Elsayed, R. Sarker, and D. Essam, "Scenario-based solution approach for uncertain resource constrained scheduling problems," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2018.
[24] Ö. H. Bettemir and R. Sonmez, "Hybrid genetic algorithm with simulated annealing for resource-constrained project scheduling," *Journal of Management in Engineering*, vol. 31, no. 5, p. 04014082, 2014.
[25] C. Fang and L. Wang, "An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem," *Computers & Operations Research*, vol. 39, no. 5, pp. 890–901, 2012.
[26] L.-Y. Tseng and S.-C. Chen, "A hybrid metaheuristic for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 175, no. 2, pp. 707–721, 2006.
[27] J. Batselier and M. Vanhoucke, "Construction and evaluation framework for a real-life project database," *International Journal of Project Management*, vol. 33, no. 3, pp. 697–710, 2015.

## APPENDIX A. MODIFIED TEST PROBLEM DATA

The data of the original test problem are taken from PSPLIB [3], and modified in terms of resource requirements of each activity. The modified resource requirements are shown in the Tables XI, XII, XIII, XIV, XV, and XVI for j11, j20, j30, j50, j75, and j100, respectively, where $j$ represents the non-dummy activity number and $r_{j,k}$, $k = 1, 2, .., K$ (e.g., $K = 2$) represents $k^{th}$ type resource demands of $j^{th}$ activity.

TABLE XI: Modified resource demands for j11 test problem

| $j$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_j$ | {3,0} | {0,3} | {4,0} | {3,0} | {0,4} | {0,4} | {0,2} | {3,0} | {3,0} | {0,5} | {4,0} |

TABLE XII: Modified resource demands for j20 test problem

| $j$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_j$ | {5,0} | {2,0} | {0,4} | {2,0} | {4,0} | {0,1} | {0,3} | {4,0} | {0,3} | {0,4} | {1,0} |
| $j$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | | |
| $r_j$ | {2,0} | {0,2} | {3,0} | {3,0} | {2,0} | {4,0} | {2,0} | {0,3} | {1,0} | | |

TABLE XIII: Modified resource demands for j30 test problem

| $j$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_j$ | {1,0} | {4,0} | {0,3} | {0,4} | {2,0} | {0,1} | {0,2} | {3,0} | {0,4} | {0,5} | {4,0} |
| $j$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $r_j$ | {2,0} | {0,3} | {4,0} | {1,0} | {3,0} | {0,2} | {0,3} | {4,0} | {0,1} | {3,0} | {4,0} |
| $j$ | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | | | |
| $r_j$ | {0,5} | {0,3} | {5,0} | {0,2} | {0,2} | {3,0} | {2,0} | {0,4} | | | |

TABLE XIV: Modified resource demands for j50 test problem

| $j$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_j$ | {5,0} | {3,0} | {0,3} | {0,2} | {3,0} | {0,2} | {0,2} | {3,0} | {0,4} | {0,5} | {4,0} |
| $j$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $r_j$ | {2,0} | {0,3} | {4,0} | {1,0} | {3,0} | {0,2} | {0,3} | {4,0} | {0,1} | {3,0} | {4,0} |
| $j$ | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| $r_j$ | {0,5} | {0,3} | {5,0} | {0,2} | {0,2} | {3,0} | {2,0} | {0,4} | {0,3} | {2,0} | {4,0} |
| $j$ | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| $r_j$ | {3,0} | {0,1} | {0,2} | {0,3} | {0,4} | {3,0} | {1,0} | {0,2} | {5,0} | {0,5} | {0,1} |
| $j$ | 46 | 47 | 48 | 49 | 50 | 51 | | | | | |
| $r_j$ | {3,0} | {2,0} | {0,4} | {2.0} | {0,1} | {4,0} | | | | | |

TABLE XV: Modified resource demands for j75 test problem

| $j$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_j$ | {1,0} | {0,3} | {2,0} | {4,0} | {3,0} | {0,4} | {0,3} | {3,0} | {0,4} | {0,5} | {4,0} |
| $j$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $r_j$ | {2,0} | {0,3} | {4,0} | {1,0} | {3,0} | {0,2} | {0,3} | {4,0} | {0,1} | {3,0} | {4,0} |
| $j$ | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| $r_j$ | {0,5} | {0,3} | {5,0} | {0,2} | {0,2} | {3,0} | {2,0} | {0,4} | {0,3} | {2,0} | {4,0} |
| $j$ | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| $r_j$ | {3,0} | {0,1} | {0,2} | {0,3} | {0,4} | {3,0} | {1,0} | {0,2} | {5,0} | {0,5} | {0,1} |
| $j$ | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 |
| $r_j$ | {3,0} | {2,0} | {0,4} | {2,0} | {0,1} | {4,0} | {0,3} | {0,4} | {1,0} | {2,0} | {0,2} |
| $j$ | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
| $r_j$ | {0,4} | {0,3} | {1,0} | {3,0} | {0,4} | {1,0} | {0,1} | {0,4} | {2,0} | {3,0} | {0,1} |
| $j$ | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | | |
| $r_j$ | {2,0} | {0,3} | {4,0} | {0,5} | {3,0} | {0,4} | {0,5} | {1,0} | {4,0} | | |

TABLE XVI: Modified resource demands for j100 test problem

| $j$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_j$ | {1,0} | {0,3} | {2,0} | {4,0} | {3,0} | {0,4} | {0,3} | {3,0} | {0,4} | {0,5} | {4,0} |
| $j$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| $r_j$ | {2,0} | {0,3} | {4,0} | {1,0} | {3,0} | {0,2} | {0,3} | {4,0} | {0,1} | {3,0} | {4,0} |
| $j$ | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| $r_j$ | {0,5} | {0,3} | {5,0} | {0,2} | {0,2} | {3,0} | {2,0} | {0,4} | {0,3} | {2,0} | {4,0} |
| $j$ | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| $r_j$ | {3,0} | {0,1} | {0,2} | {0,3} | {0,4} | {3,0} | {1,0} | {0,2} | {5,0} | {0,5} | {0,1} |
| $j$ | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 |
| $r_j$ | {3,0} | {2,0} | {0,4} | {2,0} | {0,1} | {4,0} | {0,3} | {0,4} | {1,0} | {2,0} | {0,2} |
| $j$ | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
| $r_j$ | {1,0} | {0,1} | {0,4} | {2,0} | {3,0} | {0,1} | {0,4} | {0,3} | {1,0} | {3,0} | {0,4} |
| $j$ | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 |
| $r_j$ | {3,0} | {0,1} | {2,0} | {0,3} | {4,0} | {0,5} | {3,0} | {0,4} | {0,5} | {1,0} | {4,0} |
| $j$ | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 |
| $r_j$ | {0,2} | {0,3} | {0,4} | {0,5} | {1,0} | {2,0} | {3,0} | {4,0} | {5,0} | {0,3} | {2,0} |
| $j$ | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |
| $r_j$ | {0,4} | {5,0} | {0,3} | {1,0} | {4,0} | {0,2} | {0,4} | {1,0} | {4,0} | {0,3} | {3,0} |
| $j$ | 101 | | | | | | | | | | |
| $r_j$ | {1,0} | | | | | | | | | | |