BPGA-EDA for the Multi-Mode Resource Constrained Project Scheduling Problem

Mayowa Ayodele, John McCall, Olivier Regnier-Coudert
Robert Gordon University
Aberdeen, Scotland
Email: {m.m.ayodele,j.mccall,o.regnier-coudert}@rgu.ac.uk

***Abstract*—The Multi-mode Resource Constrained Project Scheduling Problem (MRCPSP) has been of research interest for over two decades. The problem is composed of two interacting sub problems: mode assignment and activity scheduling. These problems cannot be solved in isolation because of the interaction that exists between them. Many evolutionary algorithms have been applied to this problem most commonly the Genetic Algorithm (GA). It has been common practice to improve the performance of the GA with some local search techniques. The Bi-population Genetic Algorithm (BPGA) is one of the most competitive GAs for solving the MRCPSP. In this paper, we improve the BPGA by hybridising it with an Estimation of Distribution Algorithm that focuses on improving how modes are generated. We also suggest improvement to the existing experimental methodology.**

I. Introduction

The Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP) is a generalisation of the well-known Resource Constrained Project Scheduling Problem (RCPSP). The RCPSP entails assigning start and finish times to activities that make up a project such that precedence and resource constraints are respected. The most common objective of this problem is to reduce the total project duration (makespan) [1]. Other objectives based on tardiness or earliness have also been considered in previous research [2]. The RCPSP captures real-world situations originating from different industries such as planning, maintenance, management and manufacturing [3]. However the RCPSP does not adequately capture the interaction between man-hours required for a job and the resources used [4]. This additional factor is considered in the formulation of the MRCPSP.

MRCPSP is considered to be more complex than the standard RCPSP. This is because, in addition to determining the order in which activities are executed (i.e activity scheduling), decision needs to be made as to how each activity will be executed (i.e mode assignment). The activity scheduling and mode assignment aspects of the MRCPSP are interrelated and cannot be solved in isolation.

Activity scheduling entails assigning priorities to activities that make up a project such that precedence constraints are respected (i.e each activity must be performed before its successor(s)). Mode assignment entails assigning a mode of execution to each activity of a project while respecting resource limits. A mode of execution is a vector of resources (renewable and non-renewable) and the corresponding duration required to completely perform an activity. Renewable resources are replenished per period of time while non-renewable resources are limited for the entire project. The modes of execution of an activity have varying resource and time requirements. A solution to the MRCPSP is an allocation of modes, start and finish times to all the activities of a project such that precedence and resource constraints are respected. This is with the overall aim of minimising makespan of the project.

Several meta-heuristics such as Scatter Search, Simulated Annealing, Particle Swarm Optimisation, Ant Colony Optimisation, Genetic Algorithms (GA) and Estimation of Distribution Algorithms (EDA) have been applied to this problem. A review of applications of meta-heuristics to the MRCPSP is presented in [5] and we will be making a lot of reference to this literature. The GA has been a frequent choice amongst researchers applying metaheuristics to MRCPSP [6], [7], [8], [9], [10], [11]. Although Scatter Search reports better solutions than other meta-heuristics on many problem sets, the best performing GAs [6], [8] are however better on larger problems [5].

Performances of algorithms particularly the GAs differ much from each other. This can be attributed to some implementation choices, one such choice is the use of local search procedures, which has been reported to significantly improve the performance of GAs [5]. GAs that use more local search procedures perform better than the ones that use few or none [5]. We seek to understand the effect of configuration choices on algorithm performance. We particularly investigate the *mode improvement* local search method presented in [8] which performs two major tasks: feasibility and makespan improvement.

Generally, local search methods have been applied to the mode assignment sub-problem questioning the suitability of the GA for this aspect of the MRCPSP. The crossover operator of the GA is limited in its ability to learn the interrelations between problem parameters [12]. We aim to improve the GA by applying EDA to the mode assignment aspect of the problem.

The rest of the paper is structured as follows. In Section 2, we present a background to our study. Here, we formulate the problem, present a brief review of GA configuration choices and motivation for a hybrid approach. In Section 3, we present our proposed solution approach. Section 4 presents the experimental configurations and parameter settings while results and analysis are described in Section 5. Section 6 presents our conclusions and suggests directions for future research.

II. Background

A. Problem Formulation

The MRCPSP can be formerly defined as follows: A project consists of a set of n activities. Every activity i is labeled from 1,...,n. Activity $i, i \in [2, n]$ has a set of predecessors $Pred_i$ which suggests that activity i cannot be performed until every predecessor $h, h \in Pred_i$ has been completed. Activity i must be performed in a mode $k \in [1, m_i]$, where m_i is the number of possible modes of i. Given that there are A renewable resources, each renewable resource r, $r \in [1, |A|]$ is available per period of time. The maximum per period availability of r is denoted by αmax_r. Apart from renewable resources, there are also B non-renewable resources that cannot be renewed but available for the entire project duration. The overall availability of the non-renewable resource l, $l \in [1, |B|]$ is denoted by βmax_l. Each mode of execution k of an activity i is composed of an integer vector of renewable resources $(\alpha_{i,k,1}, ..., \alpha_{i,k,|A|})$, an integer vector of non-renewable resources $(\beta_{i,k,1}, ..., \beta_{i,k,|B|})$ and the associated duration/execution time $t_{i,k}$.

The aim of the MRCPSP is to select exactly one mode of execution for each activity subject to resource and precedence constraints. This is such that makespan is minimised. We formulate the MRCPSP as follow.

Minimise ft_n subject to:

$$\forall\, i \in [1, n],\ st_i \geq ft_h\ \forall\, h \in Pred_i \tag{1}$$

Let C_p be the set of activities being executed during time period [p-1,p], then

$$\textstyle\sum_{i \in C_p} \alpha_{i,k_i,r} \leq \alpha max_r\ \forall\, r, r \in [1, |A|], \forall\, p \tag{2}$$

$$\sum_{i=1}^{n} \beta_{i,k_i,l} \leq \beta max_l\ \forall\, l, l \in [1, |B|] \tag{3}$$

We denote the start and finish times of activity i by st_i and ft_i respectively. The precedence constraint is presented in (1) while the renewable and non-renewable resource constraints are respectively presented in (2) and (3). In (2) and (3), k_i is the allocated mode of i and can only be one of the predefined modes of i. Also, $\alpha_{i,k_i,r}$ and $\beta_{i,k_i,l}$ are respectively the amount of renewable resource r and non-renewable resource l required by activity i performed in mode k_i.

B. Algorithm Configuration for Leading GAs

Genetic algorithms are amongst the most competitive algorithms for solving the MRCPSP. Results in [5] show that GAs are better on common large problem sets. The Bi-Population Genetic Algorithm (BPGA) [8] performs better than other GAs [5]. Hence, we consider it state of the art. In this section, we describe the conventional configuration choices in the most competitive GAs at solving the MRCPSP. This is with particular attention to local search methods.

The preprocessing technique of Sprecher et al. [13] is commonly executed before applying GAs to the MRCPSP. This technique which is not only limited to GA applications is formerly defined in [13]. It reduces the search space of feasible solutions by eliminating non-executable modes, redundant resources and inefficient modes. The preprocessing technique has been used in [6], [8], [7] amongst others.

Furthermore, the use of local search procedures has also been very common when solving the MRCPSP with meta-heuristics. Four main local search procedures are commonly used: *improvement of initial population, makespan improvement, feasibility improvement* and *forward-backward* procedures, these are described in [5]. These local search methods have mainly been applied to mode solutions. Their effects range from improving feasibility to improving makepsan of a solution by refining its mode selection.

The generation of initial population of solutions is often done with a quality improvement procedure: *improvement of initial population* local search. This improves the feasibility of a mode solution by changing the mode selections of randomly selected activities until feasibility is attained or a number of iterations have been reached [8], [6] .

The *Schedule Generation Scheme (SGS)* is the principal procedure for most heuristic solution to a project scheduling problem [14]. This is a step-wise procedure that builds a schedule from a schedule representation by activity incrementation (serial) or time incrementation (parallel) [14]. The parallel SGS is however less common because it is sometimes unable to reach optimal [15]. The SGS has been extended to the MRCPSP and is the conventional approach for generating schedules. Furthermore, some extensions of the SGS have been proposed which mainly combine SGS with local search.

In this paper, we investigate the *extended SGS* [8] which combines SGS with a mode improvement local search. The mode improvement procedure is a combination of *makespan improvement* and *feasibility improvement* and is described as follows. For an activity i, another mode $m_i{}'$ is selected. The Excess Resource Requirement ERR of the new mode solution x' is evaluated by summing the additional non-renewable resource requirements of x'. $ERR(x')$ is compared with that of the original mode solution x. If $ERR(x)$ is not better than $ERR(x')$, the procedure further checks for improvement in the finish time of that activity. If an improvement is achieved in the finish time of activity i, mode solution x is set to x'. Otherwise, mode solution x is retained. The pseudocode for the mode improvement procedure is presented in [8]. This procedure is reported to significantly improve the SGS.

Another extension of the SGS is the forward-backward SGS proposed in [6] which combines the *forward-backward* local search and the SGS. The forward-backward SGS seeks to improve mode selection during scheduling and iteratively transforms left justified schedules (jobs are scheduled as early as possible) into right justified schedules (jobs are scheduled as late as possible) and vice versa until no further improvement can be made in the makespan. The concept of scheduling jobs forward and backward has also been used in [8]. They proposed the BPGA which is a GA with two populations, one is a population of left justified schedules while the other is a population of right justified schedules. The GAs presented in [6] and [8] are considered the most competitive GA applications for the MRCPSP [5].

In this paper, we focus on the mode improvement local search of the BPGA.

C. Motivation for Hybridising GA with EDA

To the best of our knowledge, there are only two applications of EDAs [16], [17] to the MRCPSP.

EDAs, different from GAs have further relied on local search methods to improve activity solutions. One is the Multi-Mode Version Permutation-Based Local Search Strategy (MPBLS) in [16]. MPBLS swaps two activities in a list alongside their corresponding modes subject to precedence constraints. Another activity based local search is the random walk local search in [17]. An improvement was made to the algorithm in [16] by introducing random walk local search for the purpose of better exploring the space of activity solutions [17]. Both implementations use the Population-Based Incremental Learning (PBIL) based on two separate probabilistic models. One is used for generating activity lists while the other is used for generating mode lists.

These applications of EDA are however not as competitive as the GAs. These applications have identified the need to explore the search space of activity solutions better which is not the case with the GA. We therefore consider the GA more suitable for generating activity solutions. Furthermore, given that the GA relies on local search to improve the way modes are generated, we attribute this to its limitation in learning the structure of a problem. The EDA is designed to tackle this kind of problem by sampling a probabilistic model of promising solutions [12]. It is able to preserve structure and learn mode selections that contribute to more promising solutions. We therefore consider it more suitable for generating mode solutions. This motivates the hybridisation of BPGA with an EDA.

III. Solution Approach

In this section, we present the BPGA-EDA which is an extension of the BPGA approach in [8]. The BPGA-EDA is the BPGA assisted by an EDA for the generation of mode solutions. We use a PBIL style which refines those presented in [17] and [16]. BPGA-EDA is configured as follows.

A. Representation

The BPGA-EDA uses the random key representation for activity solutions as presented in the BPGA [8]. Each RK value serves as a priority value for the activity defined by that index. Activity with a lower priority value is considered before that with a higher value. The mode assignment is represented as a string of integers and the value at each index defines the mode in which the activity defined by that index will be performed. The string lengths correspond to the number of activities in the project.

B. Initial Population

The BPGA-EDA generates its initial population of mode and activity solutions at random. The *improvement of initial population* local search is afterward applied to mode solutions in the same way as the BPGA [8].

C. Fitness Computation

This is a measure of the quality of a solution. As much as makespan is a good discriminatory factor, it cannot be directly used as the fitness function because resource infeasible solutions will eliminate good solutions in the search. We use the fitness function proposed in [7] which is the one that has been used for the BPGA. The fitness function is as follows:

$$f(x) = \begin{cases} max\_mak\_feas\_pop + mak(x) - min\_CP + ERR(x) & if ERR(x) > 0 \\ mak(x) & otherwise \end{cases} \tag{4}$$

The fitness of a feasible solution is equal to its makespan. However, the makespan of infeasible solutions are penalised. The maximum makespan amongst the feasible solutions in a population is denoted by $max\_mak\_feas\_pop$. Also, the minimal critical path obtained by selecting modes with the least duration and assuming there is no resource restriction is denoted by min_CP. The difference between $max\_mak\_feas\_pop$ and min_CP as well as $ERR(x)$ are added to the makespan of an infeasible solution.

D. Probabilistic Model

To create a mode solution, we sample a probability matrix. The probability matrix is defined as follows.

$$M_p = \begin{pmatrix} p_{11} & \cdots & p_{1m} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nm} \end{pmatrix} \tag{5}$$

Each probability value p_{ik} in this n by m probability matrix is the probability that activity i will be performed in mode k. Modes of an activity that are impossible or have been eliminated during the preprocessing stage will have probability scores equal to 0. This is because they will not have occurred in the initial population as we execute the preprocessing procedure before the initial population is generated.

The scatter search procedure in [18] which is one of the most competitive meta-heuristic for solving the MRCPSP [5] used Sum of Durations SUD to select modes for the first population. This is a measure of mode solutions only and has been reported to have a strong correlation with fitness [18]. It is calculated as follows.

$$SUD = \sum_{i=1}^{n} t_{i,k} \tag{6}$$

The execution time $t_{i,k}$ for each activity in its allocated mode is summed up to obtain the SUD value. This approach improved the quality of their initial solutions and we initialise our EDA in a similar way.

To create the initial probability matrix, each solution in the initial population is evaluated using the SUD. This means that in our fitness function (for the first population), the makespan is replaced with the SUD value and infeasible solutions are penalised in the same way as the makespan. We rank all mode solutions in the population from best (lowest fitness value) to least (highest fitness value). The top (best) b solutions are selected from the population using truncation selection. The probability score is calculated as follows. For an activity i, we divide the number of occurrences of each possible mode of execution by b. e.g if the truncation size is ten and amongst the best ten solutions, activity 2 was executed in mode 1 : six times, mode 2 : four times and no occurrence of mode 3. The probability values p_{21}, p_{22} and p_{23} will be 0.6, 0.4 and 0 respectively.

Initialisation based on SUD is an improvement on previous approaches [17], [16] that initialise their model using equal probabilities. The disadvantage of the previous approach is the possibility of sampling impossible modes, to tackle this problem an additional step of setting impossible modes to 0 is performed. We however do not require this additional step.

Subsequently, the mode solutions are paired with activity solutions to form complete solutions. This is done in order of solution generation. These solutions are then ranked according to their fitness (based on makespan). Probability scores are recalculated for each mode of every activity. Since we use a PBIL, we update the model using a learning rate lr as shown in (7).

$$p_{ik}(g) = (lr * p_{ik}(g)) + ((1 - lr) * p_{ik}(g - 1)) \qquad (7)$$

In (7), $p_{ik}(g)$ and $p_{ik}(g-1)$ are p_{ik} values at generations g and $g-1$ respectively. The probabilistic model is updated at the end each generation until the stopping criteria is met. This is because it is more computationally efficient than updating after the creation of each solution.

E. BPGA-EDA Workflow

The BPGA approach involves the use of two separate populations POP_L and POP_R. POP_L is a population of left justified schedules while POP_R is a population of right justified schedules. In a left-justified schedule, activities are scheduled using the SGS (activities are scheduled as early as possible) while in a right-justified schedule, activities are scheduled using the backward SGS (activities are scheduled as late as possible).

In the BPGA, each individual i in POP_L or POP_R goes through crossover and serves as the first parent ($parent1$), the second parent ($parent2$) is selected by tournament selection of size two. Two offspring are generated via crossover. The activity and mode solutions of the offspring go through mutation at specified rates. The best of the two offspring from POP_L replaces individual i in POP_R and vice versa. The best solutions in POP_L or POP_R are however not replaced except the new offspring is better. The overall best solution is returned at the end of each generation. The algorithm is stopped once optimal or maximum number of schedule is reached.

In the BPGA-EDA, parameter: $edar \in [0, 1]$ is introduced to determine the rate at which EDA will be applied for generating mode solutions. We use the following notation BPGA-EDA$_{edar}$ to express the type of BPGA-EDA used. BPGA-EDA$_0$ is equivalent to the BPGA (i.e when $edar = 0$, EDA is not used), but BPGA-EDA$_1$ indicates that all mode solutions are generated by the EDA.

In the algorithm, we build two models M_{p1} and M_{p2} based on POP_R and POP_L respectively. We use the same crossover approach for RKs presented in the BPGA to generate activity solutions from $parent1$ and $parent2$. We generate mode solution for POP_R and POP_L by sampling M_{p1} and M_{p2} respectively.

The BPGA-EDA is formerly defined as follows.

```
1:  execute preprocessing procedure
2:  generate initial population $POP_L$
3:  compute fitness for each individual in $POP_L$
4:  **repeat**
5:    **if** $edar > 0$ **then**
6:      select best $b < |POP_L|$ solutions to form $S$.
7:      build probabilistic model $M_p$ from mode assignments
        of solutions in $S$
8:      initialise probabilistic models $M_{p1}$ and $M_{p2}$ with $M_p$
9:    **end if**
10:   **for** $i$ = 1 to $|POP_L|$ **do**
11:     set individual $i$ in $POP_L$ as $parent1$
12:     generate parent2 by tournament selection on $POP_L$
13:     **if** $rand < edar$ **then**
14:       perform crossover to generate two offspring activity solutions
15:       sample $M_{p1}$ to produce two offspring mode solutions
16:       generate two offspring solutions by combining
          each pair of offspring activity and mode solutions
17:     **else**
18:       perform crossover to generate two offspring solutions
19:     **end if**
20:     perform mode mutation
21:     perform activity mutation
22:     apply backward SGS to the two offspring
23:     update $POP_R$ with the best offspring
24:   **end for**
25:   **for** $i$ = 1 to $|POP_R|$ **do**
26:     set individual $i$ in $POP_R$ as parent1
27:     generate $parent2$ by tournament selection on $POP_R$
28:     **if** $rand < edar$ **then**
29:       perform crossover to generate two offspring activity solutions
30:       sample $M_{p2}$ to produce two offspring mode solutions
31:       generate two offspring solutions by combining
          each pair of offspring activity and mode solutions
32:     **else**
33:       perform crossover to generate two offspring solutions
34:     **end if**
35:     perform mode mutation
36:     perform activity mutation
37:     apply SGS to the two offspring
38:     update $POP_L$ with the best offspring
39:   **end for**
40:   **if** $edar > 0$ **then**
41:     update $M_{p1}$ using $POP_R$
42:     update $M_{p2}$ using $POP_L$
43:   **end if**
44: **until** stopping criteria satisfied
```

```
45: return overall best solution
```

Note that $rand$ is a random number between 0 and 1.

IV. Experiments

One of the principal factors in assessing the performance of an algorithm is the measure of performance. Although the meta-heuristics applied to the MRCPSP are non-deterministic, previous literature do not give information about variance. In this paper, we investigate variance as part of the measure of performance. We also describe the choice of problem set, a proposed sampling method and the measure of complexity used for parameterising BPGA-EDA.

A. Benchmark Problems

There are many benchmark problems in previous research such as Boctor [19], PSPLIB [20] and the recently designed MMLIB [5] problem sets. In this paper, we use the J10, J20 and J30 problem sets from the well-known PSPLIB available at http://www.om-db.wi.tum.de/psplib/. We chose these three as they are the most common in literature, hence useful for comparison. However, not all instances of these problem sets have at least one feasible solution, we therefore exclude them in our computation as conventionally done. After eliminating such problems, the J10, J20 and J30 have 536, 554 and 552 instances respectively.

B. Stopping Criterion and Performance Measure

When applying an algorithm to these problem sets, previous researchers have used the number of schedules as the stopping criterion. This is often set to 5000 number of schedules. The maximum CPU time is another criterion that has been used in the past. For ease of comparison, we use the maximum number of schedules as this is most commonly used. We note that a schedule refers to a single time (start and finish) assignment for each activity of a project. However, some local search methods like the makespan improvement [6], [8] may require more than one time assignment for an activity. To cater for this, Lova et al. [6] calculate the number of schedules by dividing the number of times the activities of a project have been assigned a start time by the total number of activities. This implies that each change in the start time of an activity contributes to a fraction of a schedule. For instance, if each activity of a project have been assigned a feasible start time twice, the number of schedule will be equal to 2. This method of calculating number of schedules have also used in [8], [5]. We use the same calculation in this paper.

The most common performance measure using number of schedules as stopping criteria is the average percentage deviation from optimal ($Ave\%.Dev.Optimal$). Where there are no optimal values, the critical path based lower bound (CPBLB) is used instead of the optimal. The CPBLB is estimated using the critical path based on the modes with the least durations. The $Ave\%.Dev.Optimal$ is calculated as follows

$$Ave\%.Dev.Optimal = \frac{\sum_{i=0}^{n}(((bestFit - optimal)/optimal)*100)}{n} \quad (8)$$

In eq. (8), $bestFit$ is the fitness of the best solution generated by the algorithm.

In this paper, we average the $Ave\%.Dev.Optimal$ over ten runs.

C. Generation of Sample Set

We have identified the non-deterministic nature of the BPGA. For instance, ten runs of our implementation of the BPGA on J10 produced $Ave\%.Dev.Optimal$ values ranging from 0.018 to 0.096 as shown in Figure 1.

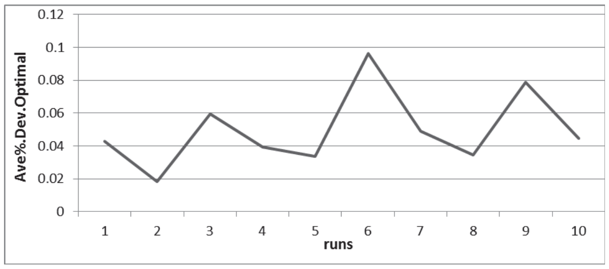

Fig. 1. Results for J10 - $Ave\%.Dev.Optimal$

The reported $Ave\%.Dev.Optimal$ value for the BPGA on J10 is 0.01. Our results are consistent with the reported BPGA performance being the best over several runs. In order to make the most meaningful comparisons amongst algorithms, in this paper we will report average performance with variance alongside best performance over several runs.

Considering the nature of the algorithm and the number of instances in the problem sets, the computational cost of experiments grow very quickly. To be able to make comparisons based on several runs and also control the computational cost, we make use of samples from the problem sets. To ensure that the samples are representative of the problem sets, we sort them by complexity and sample uniformly across the distribution. This leads us to the choice of the measure of complexity. A description of the proposed measure of complexity and sample generation technique is as follows.

Relative Resource Availability: The measures of complexity that exists in literature include the order strength, resource factor and resource strength. The order strength which is the number of precedence relationship in the problem is not only constant across each problem set but measures only the complexity of the activity scheduling aspect of the problem. Also, the resource factor and resource strength are not specific to mode generation. Since our focus is on the generation of highly fit mode assignments, we proposed a measure that relates to the ease of generating mode feasible solutions. Also, this approach takes the preprocessing technique into consideration as we only calculate the complexity after executing the preprocessing technique. Modes or constraints that are eliminated during preprocessing are therefore not taken into consideration. This means that instances with redundant non-renewable resources will have a complexity score of 0 . To generate the RRA score, we used the following formula.

$$RRA = Max\left(\frac{\sum_{i=1}^{n}\frac{\sum_{k=1}^{m_i}\beta_{i,k,1}}{|m_i|}}{\beta max_1}, \cdots, \frac{\sum_{i=1}^{n}\frac{\sum_{k=1}^{m_i}\beta_{i,k,|B|}}{|m_i|}}{\beta max_{|B|}}\right) \quad (9)$$

For a mode solution to be feasible, the sum of each non-renewable resource utilisation for all activities must be less than the maximum availability of that resource. The higher the ratio of the utilisation to availability the more difficult it is to generate a feasible solution. As shown in eq. (9), we generate a RRA score by calculating the average resource requirement of each activity for all its possible modes of execution. We sum this for all activities and divide it by the maximum non-renewable resource requirement. We do this for each non-renewable resource and pick the maximum of the values. This is because a problem is as complex as the most constrained resource.

Sampling Approach: To generate sample set of size n, the aim is to divide the problem instances in a problem set into n complexity groups and sample one problem from each group.

```
1:  order problem set $dst$ in ascending order of $RRA$
2:  initialise the required sample size $n$
3:  create an array $dataGroupSizes$ of size $n$
4:  $leftOver = |dst| \% n$
5:  for $i$ = 1 to $|dst|$ do
6:      $dataGroupSizes[i]$ = $|dst|/n$
7:      if $leftOver > 0$ then
8:          increment $dataGroupSizes[i]$
9:          decrement $leftOver$
10:     end if
11: end for
12: Define a sample set $sst$
13: Define a variable $j = 0$
14: for $i = 1$ to $n$ do
15:     Add $dataGroupSizes[i]$ to $j$
16:     Add the $j_{th}$ element in $dst$ to $sst$
17: end for
```

We determine the sizes $dataGroupSizes$ of each group by dividing the number of instances in a problem sets by sample size n. We distribute the remainder $leftOver$ over the earlier groups. This is so that groups of easier problems are the ones that get the bigger group sizes where n does not divide $|dst|$ without remainders. We do this to ensure that we are not parameterising our algorithm based on more simpler problems than harder ones. After determining $dataGroupSizes$, we order all the problems in a given problem set dst in ascending order of RRA. We use the cumulative values j of $dataGroupSizes$ to determine the problem to sample. This way we are sampling the last problem of each group and this approach ensures we do not leave the hardest problem out.

In this paper, we set the number of instances n in the sample set to 20. For example, we create J10 sample with 20 instances from J10. The complexity distribution of the J10 is shown in Figure 2. The dots along the complexity distribution line of Figure 2 indicates the sampled problems. This approach has particularly been used for parameterising the proposed algorithm.

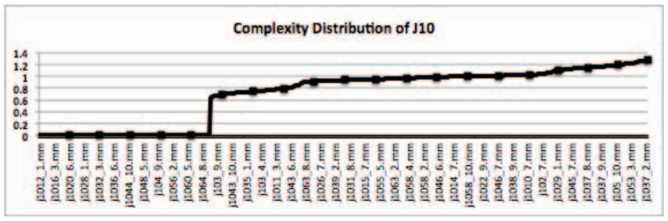

Fig. 2. Sampling along the complexity distribution of J10

D. Parameter Settings

We use all the recommended parameters of the BPGA: one-point crossover, 0.3 mode improvement rate, 0.04 and 0.02 activity and mode mutation rates respectively. The population size is calculated by $e^{3.551+\frac{22.72}{|N|}}$, all as recommended in [8]. We however note that this may not be the best set of parameters for the EDA aspect of the BPGA-EDA but we retain them for simplicity of parameter tuning. Also, difference in performance of the BPGA and BPGA-EDA may be attributed to new choice of parameters rather than the algorithmic approach.

Furthermore, the BPGA-EDA requires three additional parameters which are learning rate, truncation size and $edar$. We vary the truncation size between 10% and 50% using a step size of 10 and learning rate from 0.1 to 1.0 using a step size of 0.1. Given that EDAs and GAs traverse the search space in different ways, there has been research on combining both algorithms [21], [22]. Apart from generating all mode solutions with EDA, we consider combining crossover (GA) and sampling of the probabilistic model (EDA) for mode generation. We therefore examine $edar$ values 0.5 and 1.0.

We run these settings on sample sets of J10, J20 and J30 twenty times and average the results. Based on the average and standard deviations, we use the Friedman ranking test [23] to select the best set of parameters. Note that when certain parameters are ranked the same, we choose the set of parameters requiring the lowest average number of schedules. We found that 10% truncation size was always ranked best. However the learning rate varied a lot. Table I shows the truncation size and learning rate for BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$.

TABLE I. BPGA-EDA PARAMETERS USING EXTENDED SGS-TRUNCATION SIZE(% OF POPULATION SIZE)/LEARNING RATES

Problem Sets	BPGA-EDA$_{0.5}$	BPGA-EDA$_1$
J10	10/0.9	10/1.0
J20	10/0.6	10/0.4
J30	10/0.7	10/0.5

Furthermore, the mode improvement local search which the BPGA depends significantly on is computationally expensive as it is applied during every schedule generation. We will also compare the BPGA with BPGA-EDA with 0 mode improvement rate (i.e standard SGS). Moreover, this will give us a clearer picture of the difference between applying the GA and EDA for the generation of mode solutions. This will also help us to determine if an improvement matching that of the mode improvement local search can be made. The learning rates and truncation sizes for the BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$ within this context are different and are presented in Table II.

TABLE II. BPGA-EDA PARAMETERS USING SGS– TRUNCATION SIZE(% OF POPULATION SIZE)/LEARNING RATES

Problem Sets	BPGA-EDA$_{0.5}$	BPGA-EDA$_1$
J10	10/1.0	10/1.0
J20	20/0.8	20/0.4
J30	20/0.5	30/0.5

V. RESULTS AND ANALYSIS

In this section, we present the results from comparing the BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$ with our implementation of the BPGA. We have shown that there are variations in multiple runs of the BPGA but results for only one run is reported in literature. For this reason, we use our implementation of the BPGA so that we can compare based on average and standard deviations over several runs. Also, for a fairer assessment, we have used the same algorithm with just an additional parameter $edar$ to determine how much of EDA is used for mode generation. This means that we are comparing based on same conditions and implementation.

TABLE III. RESULTS BASED ON SGS - AVERAGE $Ave\%.Dev.Optimal$ (STANDARD DEVIATION) OF TEN RUNS

Problem sets	BPGA	BPGA-EDA$_{0.5}$	BPGA-EDA$_1$
J10	0.61 (0.08)	**0.19 (0.05)**	**0.20 (0.04)**
J20	2.34 (0.05)	**1.21 (0.06)**	**1.60 (0.07)**
J30	17.89 (0.18)	**14.67 (0.06)**	**15.06 (0.07)**

TABLE IV. RESULTS BASED ON EXTENDED SGS - AVERAGE $Ave\%.Dev.Optimal$ (STANDARD DEVIATION) OF TEN RUNS

Problem sets	BPGA	BPGA-EDA$_{0.5}$	BPGA-EDA$_1$
J10	0.05 (0.02)	**0.03 (0.02)**	**0.03 (0.02)**
J20	0.88 (0.04)	**0.53 (0.04)**	**0.69 (0.04)**
J30	14.41 (0.05)	**13.68 (0.03)**	**13.87 (0.07)**

In Tables III and IV, we present for each problem set, the average of $Ave\%.Dev.Optimal$ and the standard deviation (in brackets) over ten runs. These results are based on our implementation of the BPGA, BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$ on the J10, J20 and J30. We compare the BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$ with the BPGA. Results that are statistically better than the BPGA are displayed in bold. In this paper, we use the student t-test and a 0.05 level of significance.

- Comparison Based on the SGS: Table III shows results using the SGS for schedule generation. The BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$ have significantly lower $Ave\%.Dev.Optimal$ than the BPGA on all the problem sets: J10, J20 and J30. We observe a significant improvement in the use of EDA for mode generation .
- Comparison Based on the Extended SGS: Table IV shows results that are based on the extended SGS. The BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$ produce statistically lower $Ave\%.Dev.Optimal$ than the BPGA on J10, J20 and J30. Again, a significant improvement is achieved by using EDA for mode generation.
- Impact of Extended SGS: SGS extended by the local search method (mode improvement) not only improves the results produced by the BPGA but also improves the results of the BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$. Inasmuch as there is a clear advantage of hybridising EDA with BPGA (BPGA-EDA), the mode improvement local search method cannot be eliminated by applying the BPGA-EDA without compromising the quality of results produced. The results in Tables III and IV therefore also show that the BPGA-EDA$_{0.5}$ and BPGA-EDA$_1$ without mode improvement are worse than the BPGA with mode improvement.
- BPGA-EDA$_{0.5}$ and BPGA-EDA$_{1.0}$: comparing the two versions of BPGA-EDA: BPGA-EDA$_{0.5}$ and BPGA-EDA$_{1.0}$, the former produces significantly better $Ave\%.Dev.Optimal$ than the later on the J20 and J30 but not significantly better on the J10. This is true when the SGS or the extended SGS is used. In general, we observed that the BPGA-EDA$_{0.5}$ performs better than the BPGA-EDA$_{1.0}$. This is based on the results shown in Tables III and IV and can be attributed to the exploration ability of the GA and the exploitation ability of EDA. This is consistent with advantages of GA-EDA hybridisation noted by other researchers [21], [22].

For the purpose of comparison with existing published values, Table V shows the BPGA-EDA's best of ten runs as well as the published value of the BPGA.

TABLE V. RESULTS BASED ON EXTENDED SGS - AVERAGE % DEVIATION FROM OPTIMUM - BEST OF TEN RUNS

Problem sets	BPGA	BPGA-EDA$_{0.5}$	BPGA-EDA$_1$
J10	0.01	0.01	0.00
J20	0.57	0.46	0.62
J30	13.75	13.73	13.61

The results in Table V are similar and it is not clear which approach is better than which. We assert that results averaged over several runs provide a fairer assessment of the algorithms.

VI. CONCLUSION AND FURTHER WORK

In this paper, we propose a hybrid algorithm: BPGA-EDA, which is a Bi-population Genetic Algorithm assisted by an EDA for the generation of mode solutions. Experiments comparing the BPGA-EDA with BPGA show that hybridisation with EDA produces significant performance improvements. We are able to conclude that EDAs are better suited for generating mode solutions and GAs are well suited for generating activity solutions.

Furthermore, we show that the mode improvement local search not only improves the BPGA but also the BPGA-EDA. We have not been able to eliminate the mode improvement local search of the BPGA by using the EDA for mode generation without compromising the quality of results produced. We however note that the BPGA-EDA based on SGS may be more competitive on larger problem sets because we observed more comparable results on the larger problem set: J30. We also note that parameters of BPGA have been retained in BPGA-EDA for ease of comparison and parameter tuning. They may not be the best set of parameters for the BPGA-EDA.

Although each problem set has instances with similar characteristics, we have demonstrated that results averaged over many runs rather than one are required to fully capture the variance in the performance of an algorithm. We recommend the use of this approach.

Finally, the fact that different algorithms may be suitable for different aspects of multi-component optimisation problems (i.e. problems that can be divided into smaller optimisation problems) makes them suitable for hybrid approaches. We recommend this area for future research.

REFERENCES

[1] M. B. Wall, "A genetic algorithm for resource-constrained scheduling," Ph.D. dissertation, Massachusetts Institute of Technology, 1996.

[2] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, no. 1, pp. 1–14, 2010.

[3] K. S. Hindi, H. Yang, and K. Fleszar, "An evolutionary algorithm for resource-constrained project scheduling," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 5, pp. 512–518, 2002.

[4] A. Drexl, R. Nissen, J. H. Patterson, and F. Salewski, "Progen/πx–an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions," *European Journal of Operational Research*, vol. 125, no. 1, pp. 59–72, 2000.

[5] V. Van Peteghem and M. Vanhoucke, "An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances," *European Journal of Operational Research*, vol. 235, no. 1, pp. 62–72, 2014.

[6] A. Lova, P. Tormos, M. Cervantes, and F. Barber, "An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes," *International Journal of Production Economics*, vol. 117, no. 2, pp. 302–316, 2009.

[7] J. Alcaraz, C. Maroto, and R. Ruiz, "Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms," *Journal of the Operational Research Society*, vol. 54, no. 6, pp. 614–626, 2003.

[8] V. V. Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, 2010.

[9] M. Mori and C. C. Tseng, "A genetic algorithm for multi-mode resource constrained project scheduling problem," *European Journal of Operational Research*, vol. 100, no. 1, pp. 134–141, 1997.

[10] S. Hartmann, "Project scheduling with multiple modes: a genetic algorithm," *Annals of Operations Research*, vol. 102, no. 1-4, pp. 111–135, 2001.

[11] J. P. Reddy, S. Kumanan, and O. K. Chetty, "Application of petri nets and a genetic algorithm to multi-mode multi-resource constrained project scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 17, no. 4, pp. 305–314, 2001.

[12] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer, 2002, vol. 2.

[13] A. Sprecher, S. Hartmann, and A. Drexl, "An exact algorithm for project scheduling with multiple modes," *Operations-Research-Spektrum*, vol. 19, no. 3, pp. 195–203, 1997.

[14] R. Kolisch and S. Hartmann, *Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis*. Springer, 1999.

[15] R. Kolisch, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation," *European Journal of Operational Research*, vol. 90, no. 2, pp. 320–333, 1996.

[16] L. Wang and C. Fang, "An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem," *Computers & Operations Research*, vol. 39, no. 2, pp. 449–460, 2012.

[17] O. S. Soliman and E. A. Elgendi, "A hybrid estimation of distribution algorithm with random walk local search for multi-mode resource-constrained project scheduling problems," *arXiv preprint arXiv:1402.5645*, 2014.

[18] V. Van Peteghem and M. Vanhoucke, "Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem," *Journal of Heuristics*, vol. 17, no. 6, pp. 705–728, 2011.

[19] F. F. Boctor, "Heuristics for scheduling projects with resource restrictions and several resource-duration modes," *The international journal of production research*, vol. 31, no. 11, pp. 2547–2558, 1993.

[20] 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.

[21] J. Pena, V. Robles, P. Larrañaga, V. Herves, F. Rosales, and M. S. Pérez, "Ga-eda: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms," in *Innovations in Applied Artificial Intelligence*. Springer, 2004, pp. 361–371.

[22] Q. Zhang, J. Sun, E. Tsang, and J. Ford, "Hybrid estimation of distribution algorithm for global optimization," *Engineering computations*, vol. 21, no. 1, pp. 91–107, 2004.

[23] E. Theodorsson-Norheim, "Friedman and quade tests: Basic computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples," *Computers in biology and medicine*, vol. 17, no. 2, pp. 85–99, 1987.