Modular Analysis and Development of a Genetic Algorithm with Standardized Representation for Resource-Constrained Project Scheduling

Ali Ahrari, Saber Elsayed, Ruhul Sarker, Daryl Essam
School of Engineering and IT
University of New South Wales
Canberra, Australia
{a.ahrari, s.elsayed, r.sarker, d.essam}@unsw.edu.au

Carlos A. Coello Coello
Departamento de Computación
CINVESTAV-IPN
Mexico City, Mexico
ccoello@cs.cinvestav.mx

***Abstract*—There has been a considerable amount of research on the development of metaheuristic methods for resource-constrained project scheduling problems. Early methods followed the building blocks and even the formulation of well-understood metaheuristic methods as well as simple but effective heuristics such as forward-backward improvement. In contrast, more recent methods employ less familiar, more complex (hybrid) metaheuristics and non-standard components and formulations. Although the former may provide better results on standard test problems, it is not easy to understand how each component has contributed to improving the results and why a deviation from well-established formulations, components and methods was necessary. This research advances our knowledge about the impact of different strategies and components of customized genetic algorithms (some of which have been proposed in this study) on the optimization results. This task is performed by developing a comprehensive genetic algorithm with several familiar and potentially effective components. A modular analysis is then performed in which one component is suppressed at a time, and the resultant performance decline is analyzed. With hindsight from the modular analysis, a simple method is suggested and the importance of each component is clarified. Thus, no further simplification can be performed without compromising efficiency. Our preliminary results reveal that this customized genetic algorithm outperforms many existing methods and can compete with the most successful ones, which, in many cases, are much more complex than our approach.**

***Index Terms*—Combinatorial optimization, evolutionary algorithm, heuristic, schedule generation scheme, self-adaptation**

I. Introduction

A resource-constrained project scheduling problem (RCPSP) aims to schedule a predefined set of activities with given durations such that the problem's objective (usually the project makespan) is optimized while the constraints on available resources and precedence relationships of activities are satisfied. Although there are exact methods that can guarantee finding the optimal solution (e.g., integer programming with the branch and bound method proposed in [1]), these methods can be applied only to small-scale problems since the RCPSP belongs to the class of NP-hard optimization problems [2]. Therefore, there has been a lot of research on the use of heuristics, and more recently, on metaheuristic methods for RCPSPs.

Although it is possible to directly optimize the start/finish times of activities (see the method in [3]), most (meta)heuristic methods optimize an encoded representation of schedules such as an activity list (integer values showing the sequence of activities) or random keys (continuous values showing the priority of activities). The start times of the activities, which is referred to as "a schedule", is then determined by applying a schedule generation scheme (SGS) to the encoded representation. An SGS creates a precedence and resource feasible schedule by stepwise extension of a partial schedule given the activity list or priorities. An SGS can be parallel or serial and may create the partial schedule from left to right (forward SGS) or right to left (backward SGS). A more detailed discussion of SGSs will be provided in Subsection II-B.

Priority rule-based heuristic methods optimize a RCPSP by defining a rule that determines the priority of activities based on one or more features of the problem, such as the latest finish time (LFT) [4]. Then, an SGS is employed to create the schedule, considering the priories of the activities. The priority rule can be static or dynamic (depending on whether the priorities of the activities change during the steps of SGS) and local or global (depending on whether only the information of the activity under consideration is used to define the priority) [2]. The resulting heuristic method can be single-pass (generating only one schedule) or multi-pass (generating more than one schedule). A single-pass method combines one SGS with one priority rule and returns a unique schedule. Multi-pass methods may employ multiple priority rules, multiple SGSs, iterative forward-backward scheduling, or some randomness in the SGS scheme or priorities [2]. More recent research on rule-based priority methods focus on the identification or even automatic formulation of new priority rules [5], [6].

This research has been funded by Australian Research Council Discovery Project DP190102637.

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

Metaheuristic methods provide an alternative to heuristic methods for optimizing RCPSPs. They generally employ a population of solutions that gradually converge to a good schedule. Their search operators involve some randomness which improves their global search capability. By the 2000s, many well-known metaheuristic methods such as genetic algorithms (GAs) had been applied to RCPSPs [7]. The superiority of metaheuristic methods over heuristic ones was soon discovered [7], [8], especially when the evaluation budget was abundant enough to reveal the advantages of learning from the past [8], which is a characteristic of metaheuristic methods. Thereafter, many metaheuristic methods, and in some cases, methods that hybridize two or more metaheuristics, have been developed and applied to RCPSPs (see [9] for a survey and comparison of hybrid methods). As a general trend, more recent metaheuristic methods outperform older ones; however, they are more complex with additional components, such as building blocks, or non-standard operators.

A. Motivation for This Work

Even in their general form, the proper formulation of a building block has a substantial impact on the performance of a metaheuristic method. For example, evolutionary algorithms, the most well-known class of metaheuristic methods, employ selection, recombination, and mutation. Moreover, when applied to RCPSPs, these methods should employ some problem-specific building blocks, such as forward-backward improvement (FBI) [10], customized representation, and SGS, to improve their capability, which further exacerbates their complexity. Non-standard modifications of metaheuristics and their hybridization also intensify this complexity.

It seems that the primary focus of many new studies has been on developing methods that can compete or even outperform the existing ones. Even for the most complex methods, the overall performance can be easily compared numerically on standard test problems (e.g., the test problems of PSPLIB [11]). Therefore, being a complex method is not a drawback from this perspective. However, there is the caveat that such studies provide little contribution to advancing knowledge in this field. In many cases, theoretical or numerical motivations for non-standard components of a method are missing or insufficient. It remains unclear whether each modification was necessary to reach better results. Even if that was the case, the question remains of whether the additional complexity was worth the gain obtained in the results (practical significance of improvement). This knowledge is particularly important for other researchers who wish to know how each component or modification has contributed to the performance of a successful method and properly integrate it with new or existing methods.

B. Objectives and Contributions

The primary objective of this research is NOT to develop a method that outperforms the best existing methods for RCPSPs but to reveal the impact of some existing and a few novel strategies and operators. It aims to pave the path for developing methods that are robust and efficient but do not have unnecessary complexities since each component provides a substantial contribution to the overall performance.

One way to identify promising components is to find the shared features among a large number of existing methods, a methodology followed by Kolisch and Hartmann in their comprehensive comparative study [8]. This study favors a more controlled methodology in which the effect of each component is studied in isolation. To achieve this task, it develops a self-adaptive GA with reasonable complexity. Justification for the formulation of each building block of this GA is provided. Controlled experiments are performed to reveal the impact of each component by comparing the performance of the developed method when that specific component is active or suppressed. In particular, this study:

- formulates a standardization procedure for the representation of schedules to avoid representation redundancy. The importance of this module is numerically demonstrated.
- explores the effect of self-adaption of the direction (forward or backward) and the type (serial or parallel) of SGS.
- analyzes the importance of crossover direction (forward or backward)
- Investigates potential merits of restrictions for recombination between parents with identical SGS type or direction.
- studies the best setting for the forward-backward improvement operator.

The rest of this paper is organized as follows: Section II provides some technical preliminaries related to this study. Section III develops a self-adaptive GA. Controlled simulations are performed in Section IV. A comparison with some existing methods is performed in Section V. Finally, our conclusions are drawn in Section VI.

II. Technical Preliminaries

Some technical preliminaries related to this study are explained in this section.

A. Problem Formulation

The RCPSPs considered in this study is summarized as follows: There are $D+2$ activities $(0, 1, 2, \ldots, D+1)$, in which activities 0 and $D+1$ are dummy activities that correspond to the start and finish time of the project, respectively [2]. Activity j takes d_j time units to complete. There are K types of resources. The number of available units for each resource is denoted by $R_k, k = 1, 2, \ldots, K$. Activity j requires r_{kj} units of the k^{th} resource type. Activities 0 and $D+1$ have zero duration and zero resource requirements.

The objective is to find the start/finish time of D non-dummy activities such that the project makespan is minimized while the schedule is precedence and resource feasible. The precedence constraints enforces that each activity can start only when all of its predecessors have finished. The set of predecessors of activity j is denoted by $\mathcal{P}_j$. The resource constraints ensure that for each type of resource, the sum of the resource units required by the ongoing activities does

not exceed the available resources. This problem can be formulated as follows:

$$\begin{aligned} &\text{Minimize } FT_{D+1} \\ &\text{Subject to:} \\ &FT_h \leq FT_j - d_j, \forall h \in \mathcal{P}_j\,, j = 1, 2, \ldots, D+1 \\ &\sum_{j \in \mathrm{A}(t)} r_{kj} \leq R_k,\ k = 0, 1, 2, \ldots, K \\ &FT_j \geq 0,\ j = 0, 1, 2, \ldots, D+1 \end{aligned} \tag{1}$$

In this formulation, FT_j is the finish time of the j^{th} activity and $\mathrm{A}(t) = \{j \in \{0, 1, \ldots, D+1\} | FT_j - d_j \leq t < FT_j\}$ is the set of activities that are ongoing (active) at time t. In this formulation, the first line states the precedence constraints, and the second line describes the resource constraints. Preemption is not allowed, which means that each activity cannot be interrupted once it has started.

B. Schedule Generation Scheme

The choice of SGS can have a major impact on optimization results. An SGS is a stepwise approach which can be serial (activity-based increment) or parallel (time-based increment) [7]. Given a partially set schedule, in which the start times of activities $0, 1, \ldots, k$ have already been set, a serial SGS finds the earliest possible time for activity $k+1$. In contrast, a parallel SGS determines the schedule by increments over time. For time t, the eligible activities (i.e., the activities whose predecessors have been completed) are determined, and the one with the highest priority would start at time t. If no eligible activity can be set, t is increased.

The schedule generated from a serial SGS is always an active schedule [12] (i.e., a schedule in which no activity can start earlier without delaying another activity [2]). For makespan minimization, the optimal schedule is always an active schedule [2]. A parallel SGS results in a non-delay schedule [13], a subset of active schedules for which even if preemption is allowed, no activity can start earlier without delaying another activity. The optimal schedule is not always a non-delay schedule; therefore, a parallel SGS has a severe theoretical shortcoming: There may be no representation of the optimal solution which is mapped to the optimal schedule if a parallel SGS is applied. This may explain why most metaheuristic methods opt for serial SGS or the union of both types of SGSs [8].

Hartmann and Kolisch [7] showed that although serial SGS can potentially find the global optimum, it may fall behind parallel SGS for large-size problems with a limited evaluation budget. They argued that parallel SGS only searches the space of non-delay schedules. Besides, a parallel SGS implies a greedy use of available resources, which on average, results in a good schedule. Thus, it can be beneficial when there is not enough evaluation budget to sufficiently explore the whole space of active schedules.

Besides the type of SGS, which can be serial or parallel, the direction of SGS may affect the optimization process [14]. The explained procedure of SGSs assumed that a schedule is generated from left to right (forward scheduling). A backward SGS generates the schedule from right to left. For example, when the partial schedule of activities $j, j+1, \ldots, D+1$ has been generated by a backward serial SGS, the latest time in which activity $j-1$ may finish is determined, and this activity is set such that it finishes at this time. A backward serial SGS results in a right-active schedule. Similarly, a backward parallel SGS can be used by decreasing the time and applying one final shift in the start times of all activities such that activity zero starts at time $t = 0$. For example, the GA in [14] generated both froward and backward schedules and selected the better one as the mapped schedule for the activity list. Another example is the GA developed in [15], which divides the population into two subpopulations: those generated by a forward SGS and those generated by a backward one. It also restricts the recombination to solutions in the same subpopulation.

C. Forward-Backward Improvement

Forward-backward improvement (FBI) [10] is a simple yet robust strategy to improve a generated schedule from SGS. In forward improvement, each activity is shifted to the earliest time that does not violate the precedence and resource constraints. Activities that start earlier are shifted first. Similarly, in backward improvement, activities are shifted to the latest time such that no constraint is violated. In this case, activities that finish later are shifted first. In the case of identical start/finish times, some tie-breaking heuristics should be used. Iterative forward and backward scheduling results in a reduction or no change in the schedule makespan. However, since each forward or backward pass is counted as a solution evaluation, the FBI process terminates when there is no change in two successive forward/backward passes. The robustness of FBI allows for its integration with almost any search method. A statistical analysis from Kolisch and Hartmann [8] showed the importance of FBI for metaheuristic-based methods.

III. Customized GA-Based Method

This study employs a self-adaptive GA as the core metaheuristic method for optimization. The structure and the building blocks of this GA are explained in this section.

A. General Structure

The building blocks of the proposed GA, which is called self-adaptive genetic algorithm with standardized activity list (SAL-SAGA), are as follows:

- Activity list representation
- Self-adaptation of SGS type and direction
- Forward and backward multipoint discrete crossover
- Precedence-compliant insertion mutation
- Tournament selection with dynamic tournament size
- FBI with controlled evaluation budget
- Standardized representation

Fig. 1 shows the flowchart of SAL-SAGA and the interaction between its building blocks. These building blocks are discussed in this section.

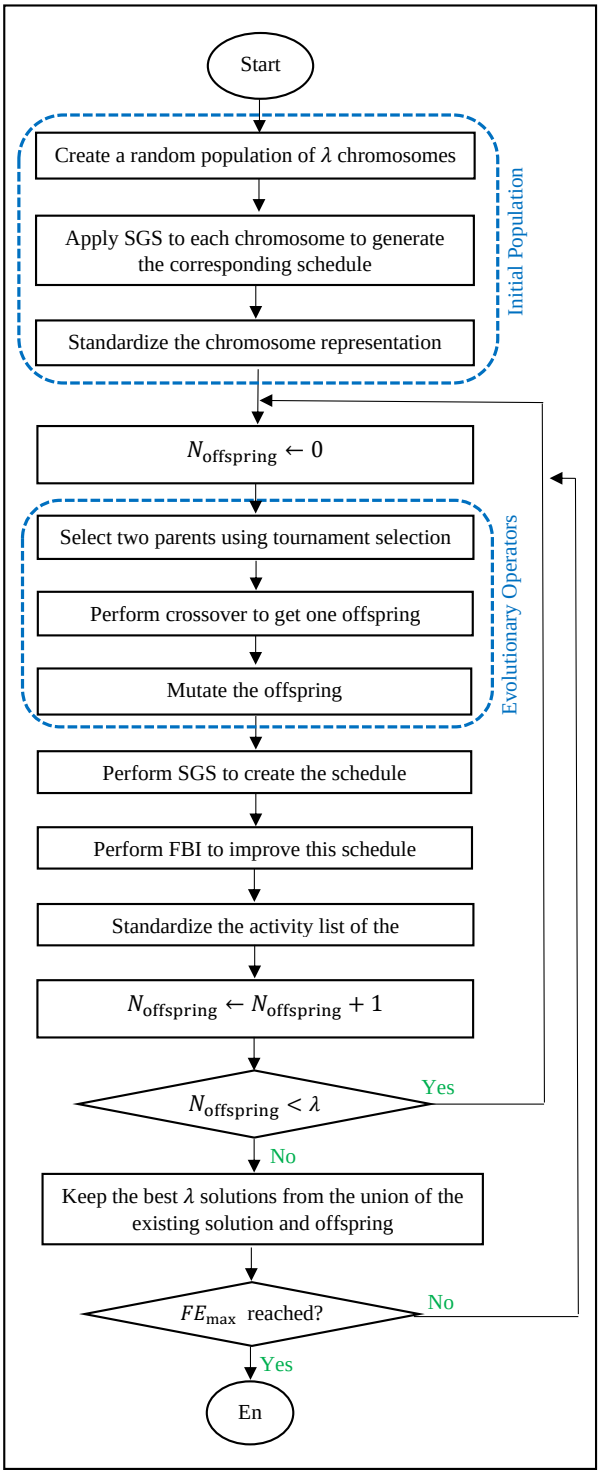

Fig. 1. Flowchart of the SAL-SAGA

B. Solution Representation

Each represented solution (encoded solution) is a chromosome of size $D + 2$. The first two genes represent the SGS type (0 for serial and 1 for parallel) and the SGS direction (0 for forward and 1 for backward). The next D genes form a precedence feasible activity list. Fig. 2 depicts

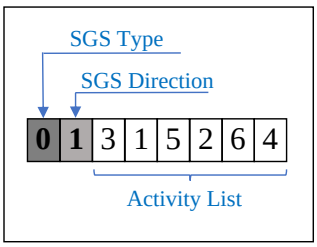

Fig. 2. A chromosome representing an encoded solution with six non-dummy activities. The values of the first two genes show that a backward serial SGS will be applied to this encoded solution to generate the corresponding schedule

an exemplary chromosome with six non-dummy activities, to which the backward serial SGS will be applied to generate the corresponding schedule.

C. Schedule Generation Scheme

The employed schedule generation scheme can be serial or parallel, forward or backward. The type and direction of the SGS are determined by the first two genes of the chromosome.

D. Forward-Backward Improvement

After generating a schedule from a chromosome, FBI is performed iteratively to improve the generated schedule. For schedule $\mathbf{S}$ with the corresponding chromosome $\mathbf{C}$, forward and backward improvements are performed alternately. Assuming c_2 is the second gene of $\mathbf{C}$ (the gene storing the SGS direction), the following rules are applied:

- If the second gene in $\mathbf{C}$ is zero, then backward improvement is performed because it shows that a forward SGS or a forward improvement pass has recently been applied, and thus, the schedule is already left-active. After performing backward improvement, the value of the second gene in the chromosome is changed to one.
- Similarly, if the second gene in $\mathbf{C}$ is one, then a forward improvement is performed because it shows a backward SGS or a backward improvement pass has recently been applied, and thus, the schedule is already right-active. After performing the forward improvement, the value of the second gene in the chromosome is changed to zero.

Each forward or backward pass counts as one iteration of FBI. A maximum of FE_{FBI} iterations of FBI are allowed per each schedule. However, the FBI terminates if there is no change in the makespan in two successive iterations.

E. Evolutionary Operators

The employed GA in this study is a (λ+λ)-GA, in which λ parents generate λ offspring. The parents for the next generation are selected from the union of the current parents and the recently generated offspring. Based on some preliminary parameter studies, we set $\lambda = \max\{2, \lfloor\sqrt{FE_{\max}}\rfloor\}$, in which $FE_{\max}$ is the evaluation budget (measured in terms of the maximum number of function evaluations). The evolutionary components of the employed GA are discussed in this section.

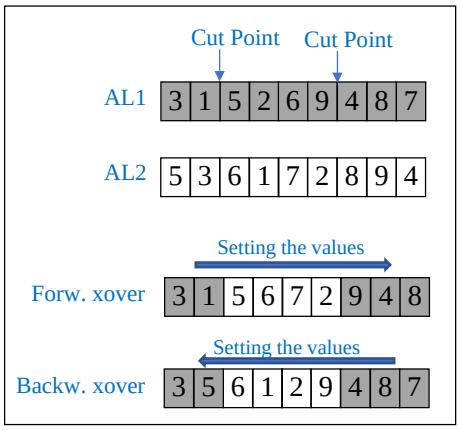

Fig. 3. Effect of the crossover direction on two parent activity lists (AL1 and AL2) and the resultant offspring

1) Crossover: The crossover operator in GAs generally creates two offspring from two parents. The crossover in this study, however, produces only one offspring from two parents. This allows for a less noisy selection especially when the population size is small.

The crossover of the first two genes is simple: the corresponding gene in the offspring is the average of the corresponding genes in the parents. If this value is 0.5, it is stochastically rounded down/up to zero or one with equal probability.

For recombination of the activity lists, this study employs a generalized case of multi-point discrete crossover developed in [16] to avoid fixing the number of cut points. This study favors the selection of the number of cut points from a truncated normal distribution with standard deviation σ_{xover} with lower and upper bounds of 1 and D. By default, $\sigma_{\text{xover}} = 1$.

It is worth noting that, like SGS, crossover can be performed from left-to-right (forward) and right-to-left (backward). For example, Alcaraz [14] used an extra gene to decide on the direction of the crossover based on whether forward or backward SGS was used for the parents. In this study, we use forward and backward crossover, each with a fixed probability of 0.5. Fig. 3 depicts one example of forward and one example of backward crossover.

2) Mutation: For the first and second gene of a chromosome, mutation is performed with a probability of $P_{\text{mut}}^{\text{Type}}$ and $P_{\text{mut}}^{\text{Dir}}$, respectively. The mutation changes the value of the gene from zero to one or vice versa. Based on some preliminary results, we set $P_{\text{mut}}^{\text{Type}} = P_{\text{mut}}^{\text{Dir}} = 0.1$.

For the activity list, a standard insertion mutation is applied. For each activity (in a random sequence), mutation is performed with a probability of $P_{\text{mut}}^{\text{AL}}$. If activity j is selected for mutation, its closest predecessor and successor in the activity list are identified, and activity j is placed in a random position between these two limits. This type of insertion keeps the precedence feasibility of the activity list. Based on a few numerical experiments, the value of $P_{\text{mut}}^{\text{AL}} = 1/(D+1)$ is used.

3) Selection: Tournament selection is employed to select parents for recombination. Quite often, the tournament size is two, which is roughly equivalent to linearly decreasing weights (e.g. in [14]). This setting is followed in this study.

It may be beneficial to restrict the recombination between chromosomes that have SGSs of identical type and direction (e.g., the GA in [15]). Therefore, the selection operator has two other Boolean parameters: r_{type}, r_{dir}, for which a true value means that the restriction is enforced on the type and direction of SGS, respectively, when selecting a pair of parents for crossover. This means that the selected parents for recombination must have identical SGS type/direction. Otherwise they are rejected and the tournaments are repeated.

F. Standardization of Representation

One challenge associated with using a representation for RCPSPs is that there is not a one-to-one correspondence between the represented solution and the generated schedule. In summary:

- There can be multiple activity lists which map to the same schedule when applying a SGS [14]. If a random keys representation is employed, there will be plateaus in the search space.
- For a fixed activity list, the generated schedule depends on the type and direction of the employed SGS. If both types of SGS (serial and parallel) and both directions (forward and backward) are employed, one activity list may be mapped to four different schedules.

The first redundancy creates extra local/global optima in the search space, all of which map to a single schedule. We speculate that this redundancy is severely detrimental because it divides the population into multiple niches, each around a different minimum, thus reducing the number of individuals that search for a basin. Recombination among solutions from different niches is unlikely to result in good solutions. It is noteworthy that this redundancy has nothing to do with multimodal optimization [17] (finding multiple solutions to a problem) since the schedules generated from all these different representations are identical.

The representation standardization module aims to overcome this challenge by using one standard activity list for all activity lists that map to identical schedules. For the schedule created from a forward SGS or a forward improvement pass, the standard activity list is the one in which activities that start earlier appear earlier in the list. If two activities have an identical start time, the one with the smaller slack time should come first. If the slack times are also identical, the activity with the smaller number comes earlier in the list.

Following the same idea, for the schedule created from a backward SGS or a backward improvement pass, the standard activity list is the one in which activities that finish later appear later in the list. If two activities have identical finish time, the one with smaller slack time should come later. If slack times are also identical, the activity with the greater number comes later in the list.

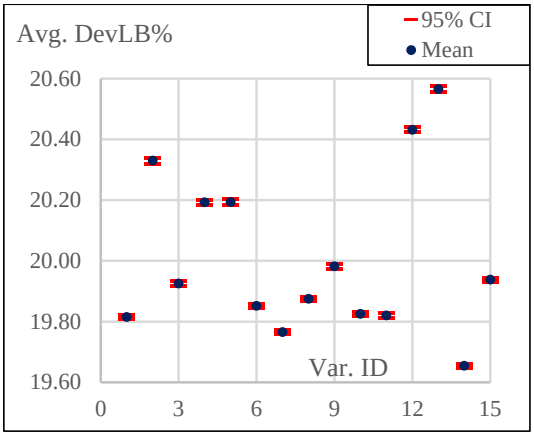

Fig. 4. Average deviation from the lower bound (mean and 95% confidence interval) calculated for 15 variants of SAL-SAGA on 480+480+600 problems when $FE_{\max} = 1000$

IV. Modular Analysis

This section performs a modular analysis to discover the impact of each component of SAL-SAGA. Each time, one component is suppressed, and the significance (statistical or practical) of the performance decline is analyzed to reveal the impact of that component. The default variant of SAL-SAGA is as follows:

- SGS: Self-adaptation of the SGS type and direction.
- Crossover: Forward and backward crossover (each one may be selected with a probability of 0.5). No restriction for similarity of SGS type or direction is enforced for crossover ($r_{\text{type}} = r_{\text{dir}} = 0$). Besides, σ_{xover}=1.
- Standardization: The activity list is standardized after performing FBI (see Fig. 1)
- FBI: $FE_{\text{FBI}} = 2$: one forward and one backward pass are performed. The order of these passes depends on the SGS direction.

In addition to this default variant, which is denoted by SAL-SAGA, some other variants are considered, each identical to SAL-SAGA except for the explained component. Table I presents these variants.

The widely adopted test problems of PSPLIB [11] are used for performance analysis. Each variant is tested on J30, J60, and J120 problems (480+480+600=1560 problems). For each problem, 20 independent runs were performed and the deviation from the lower bound was calculated:

$$DevLB = \frac{makespan_{\min} - LB}{LB} * 100\% \qquad (2)$$

in which LB is the lower bound calculated using the critical path method and $makespan_{\min}$ is the minimum makespan found by the method. Fig. 4 shows the mean and 95% confidence interval of $DevLb$, when calculated over 1560 problems. From the obtained results, the following observations can be made:

- *Effect of SGS type*: Using only serial or parallel SGS results in a significant decline in performance (compare Variant 1 with Variants 2 and 3). Using both with a fixed probability of 0.5 for each (row 6) is still worse than self-adaptation of the SGS type, although it is better than using only one type of SGS. This clarifies the usefulness of (self-)adaptation of the SGS type. If only one SGS type must be used, parallel SGS turns out to be a better choice for J60 and J120, whereas serial SGS is more successful for J30. Considering the limited evaluations budget, this observation parallels the findings in [7] in which parallel SGS may work better in the short term for problems of higher dimensionality.
- *Effect of SGS Direction*: Using self-adaptation for the SGS direction results in a significant improvement compared to using only one direction for SGS (row 1 versus rows 4 and 5). Surprisingly, using both forward and backward SGS with a fixed probability of 0.5 (row 7) is even a better or at least an equally good option. This shows that self-adaptation has not worked for the SGS direction. We speculate that the offspring of parents that were generated/improved in the forward (backward) direction are more likely to be good solutions if they are scheduled in the backward (forward) direction.
- *Effect of restrictions for crossover*: Restricting recombination to parents with identical SGS type or direction is detrimental to the algorithm's performance (row 1 versus rows 8 and 9). Besides, it introduces an additional operator to the method.
- *Effect of crossover direction*: Using both forward and backward crossover with a fixed probability of 0.5 for each of them does not show a practical advantage over using only forward or only backward crossover.
  Effect of standardization of activity list: Performing standardization of the represented activity list resulted in a substantial improvement (row 1 versus row 12).
  FBI effect: Suppressing FBI results in a major decline in performance (row 1 versus row 15). Using FBI for only one iteration (forward pass if the schedule was generated by backward SGS, backward pass otherwise) is a better choice than the conventional choice of $FE_{\text{budget}} = 2$. An additional increase in $\text{FE}_{\text{budget}}$ results in a minor deterioration in performance since the FBI may be terminated after two iterations if there is no further improvement.

V. Comparison with Other Methods

Following the hindsight from Section IV, the recommended variant of SAL-SAGA is the default one except that:

- Forward and backward SGS are performed with a fixed probability of 0.5 for each (instead of the self-adaptation of the SGS direction).
- The budget of FBI is one ($FE_{\text{FBI}} = 1$).

The test problems of PSPLIB [11] have generally been optimized with budgets of 1000, 5000, and 50,000 evaluations. The same setting is followed in this study. Table II compares the results of the suggested variant of SAL-SAGA with some of the existing methods for RCPSPs. For J30, the results are the average difference from the theoretical optimum. For J60

TABLE I
DESCRIPTION OF DIFFERENT VARIANTS OF SAL-SAGA FOR MODULAR ANALYSIS

Var. No.	Variant	Description
1	SAL-SAGA	Default variant
2	SAL-SAGA-{S-SGS}	Only serial SGS for schedule generation is used
3	SAL-SAGA-{P-SGS}	Only parallel SGS for schedule generation is used
4	SAL-SAGA-{F-SGS}	Only forward SGS for schedule generation is used
5	SAL-SAGA-{B-SGS}	Only backward SGS for schedule generation is used
6	SAL-SAGA-{S&P-SGS}	Serial or Parallel SGS is used with equal probabilities (no self-adaptation)
7	SAL-SAGA-{F&B-SGS}	Forward or backward SGS is used with equal probabilities (no self-adaptation)
8	SAL-SAGA-{Xover-Type}	Crossover is restricted between parents who have identical SGS type
9	SAL-SAGA-{Xover-Dir}	Crossover is restricted between parents who have identical SGS Direction
10	SAL-SAGA-{Xover-F}	Crossover is performed only in forward direction
11	SAL-SAGA-{Xover-B}	Crossover is performed only in backward direction
12	SAL-SAGA-{Standard}	Standardization of the activity list after FBI is suppressed
13	SAL-SAGA-{$FE_{\text{FBI}} = 0$}	FBI is suppressed
14	SAL-SAGA-{$FE_{\text{FBI}} = 1$}	FBI is performed for one iteration only (either forward or backward improvement)
15	SAL-SAGA-{$FE_{\text{FBI}} = 4$}	FBI is performed with a maximum of four iterations. It terminates if there is no change in the last two iterations

and J120, the performance indicator is the average deviation (percentage) from the lower bound determined by the critical path method (see equation (2)).

A comparison of the results reveals that SAL-SAGA outperforms or is at least competitive with respect to the majority of these methods, especially when the evaluation budget is limited. More importantly, SAL-SAGA consists of simple and intuitively familiar components, and the effect of each of them has been well-understood from the literature and from this study.

VI. SUMMARY AND CONCLUSIONS

Providing promising results on standard test problems is indeed an essential factor for the evaluation of optimization methods. However, it is also important that these methods are *created as simple as possible, but not simpler*.[1] For metaheuristic methods, this implies not only fewer and simpler components but also more occasional deviations from well-understood strategies and formulations. Of course, additional complexity can be introduced if the importance of a component is properly justified and demonstrated. A modular analysis has been suggested and followed in this study to reveal the significance (statistical and practical) of extra operators. This methodology analyzes the performance decline when the specific component has been suppressed.

The simplicity of a method and clarification of the actual importance of each module can substantially contribute to advancing knowledge in the field of RCPSPs. Other researchers can easily understand how each component of a particular method has contributed to its success. This knowledge allows them to excerpt a specific module as well as to integrate it with other (new or existing) algorithms to introduce more robust and efficient methods.

Following these objectives, this study has developed a comprehensive GA that integrates several components. The underlying ideas of these components are well-understood, and for many of them, there are comprehensive numerical or theoretical analyses available in the literature. This GA has allowed us to investigate the potential benefit from each module using the aforementioned modular analysis. In particular, our modular analysis revealed that:

- Standardizing the representation of solutions is crucial. For a fixed SGS, it ensures a one-to-one correspondence between the represented solution and the generated schedule. Thus it eliminates the formation of redundant basins in the search space of solutions. This study has developed a standardized representation for the activity list. However this idea can be easily applied to other types of representations.
- Using both types (serial and parallel) and both directions (forward and backward) with SGS is advantageous. For the SGS type, self-adaptation works well. In contrast, for the SGS direction, selecting the SGS direction with predefined probabilities worked better than self-adaptation.
- When employing different SGS types or directions, there is no need to ensure that only parents with identical SGS type or direction may recombine. In fact, applying such restrictions turned out to be slightly disadvantageous.
- FBI has a substantial impact on performance. In our method, FBI with a budget of one iteration was the best option. This implies one forward (backward) pass for schedules generated from a backward (forward) SGS or a previous backward (forward) pass. It is noteworthy that performing only one iteration of FBI has a similar effect to changing the direction of SGS. Therefore, the suitability of this setting (FBI with one iteration) may be valid only for methods that use both directions for SGS.

In hindsight of this analysis, a slightly simpler GA was proposed and compared with several existing (complex) methods on standard test problems given a limited evaluation budget. This GA could outperform many of these methods and could compete with the rest except for a few. More importantly, this GA consists of distinguishable and well-justified modules, the importance of which has been already shown. Thus, no further simplification could be made without sacrificing the quality of

[1]From a quote by Albert Einstein.

TABLE II

COMPARISON OF THE RESULTS OF THE SUGGESTED VARIANT OF SAL-SAGA WITH THE RESULTS OF SOME EXISTING METHODS FOR DIFFERENT VALUES OF $FE_{\max}$. FOR J30, THE RESULTS ARE THE AVERAGE DIFFERENCE FROM THE THEORETICAL OPTIMUM. FOR J60 AND J120, THE PERFORMANCE INDICATOR IS THE AVERAGE DEVIATION (PERCENTAGE) FROM THE LOWER BOUND AS DETERMINED BY THE CRITICAL PATH METHOD.

Method	Year	Results From	$FE_{\max} = 1000$			$FE_{\max} = 5000$			$FE_{\max} = 50,000$		
			J30	J60	J120	J30	J60	J120	J30	J60	J120
SAL-SAGA (This work)	2021	-	0.1	11.36	34.14	0.05	10.95	32.72	0.01	10.71	31.24
TS-MODE [18]	2020	[18]	0.06	11.9	34.41	0.01	11.21	32.86	0	10.63	30.59
Sequential(SS(FBI)) [19]	2018	[18]	0.1	11.38	34.01	0.02	10.93	32.52	0	10.58	31.16
COA [20]	2017	[20]	0.04	11.13	34.04	0	10.77	32.9	0	10.58	31.22
PSO-HH [21]	2014	[20]	0.26	11.74	35.2	0.04	11.13	32.59	0.01	10.68	31.23
GA-MBX(FBI) [22]	2013	[18]	0.14	11.33	34.02	0.04	10.94	32.89	0	10.65	31.3
GA(LS) [23]	2011	[18]	1.83	11.35	33.45	1.27	10.53	31.51	0.71	10.52	30.45
Scatter Search—FBI [24]	2006	[8]	0.27	11.73	35.22	0.11	11.1	33.1	0.01	10.71	31.57
GA-FBI [25]	2005	[8]	0.34	12.21	35.39	0.2	11.27	33.24	0.02	10.74	31.58
GA—self-adapting [26]	2002	[8]	0.38	12.21	37.19	0.22	11.7	35.39	0.08	11.21	33.21
GA—forw.–backw. [14]	2001	[8]	0.33	12.57	39.36	0.12	11.86	36.57	NA	NA	NA

the results. It is predicted that the insight provided from our modular analysis paves the way to the development of simpler and more efficient methods for RCPSPs.

VII. ACKNOWLEDGMENT

The research is supported by a Australian Research Council project ARC DP190102637. Computational work in this project was supported by Australian National Computational Infrastructure. The last author acknowledges support from CONACyT grant no. 1920 and from a SEP-Cinvestav grant (application no. 4).

REFERENCES

[1] P. Brucker, S. Knust, A. Schoo, and O. Thiele, "A branch and bound algorithm for the resource-constrained project scheduling problem," *European journal of operational research*, vol. 107, no. 2, pp. 272–288, 1998.

[2] R. Kolisch and S. Hartmann, "Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis," in *Project scheduling*. Springer, 1999, pp. 147–178.

[3] Y. C. Toklu, "Application of genetic algorithms to construction scheduling with or without resource constraints," *Canadian Journal of Civil Engineering*, vol. 29, no. 3, pp. 421–429, 2002.

[4] E. W. Davis and J. H. Patterson, "A comparison of heuristic and optimum solutions in resource-constrained project scheduling," *Management science*, vol. 21, no. 8, pp. 944–955, 1975.

[5] W. Guo, M. Vanhoucke, J. Coelho, and J. Luo, "Automatic detection of the best performing priority rule for the resource-constrained project scheduling problem," *Expert Systems with Applications*, p. 114116, 2020.

[6] S. Chand, Q. Huynh, H. Singh, T. Ray, and M. Wagner, "On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems," *Information Sciences*, vol. 432, pp. 146–163, 2018.

[7] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 127, no. 2, pp. 394–407, 2000.

[8] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *European journal of operational research*, vol. 174, no. 1, pp. 23–37, 2006.

[9] R. Pellerin, N. Perrier, and F. Berthaut, "A survey of hybrid metaheuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 280, no. 2, pp. 395–416, 2020.

[10] K. Y. Li and R. J. Willis, "An iterative scheduling technique for resource-constrained project scheduling," *European Journal of Operational Research*, vol. 56, no. 3, pp. 370–379, 1992.

[11] 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.

[12] R. Kolisch, "Efficient priority rules for the resource-constrained project scheduling problem," *Journal of Operations Management*, vol. 14, no. 3, pp. 179–192, 1996.

[13] ——, "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.

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

[15] D. Debels and M. Vanhoucke, "A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem," *Operations Research*, vol. 55, no. 3, pp. 457–469, 2007.

[16] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Research Logistics (NRL)*, vol. 45, no. 7, pp. 733–750, 1998.

[17] S. Das, S. Maity, B.-Y. Qu, and P. N. Suganthan, "Real-parameter evolutionary multimodal optimization—a survey of the state-of-the-art," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 71–88, 2011.

[18] K. M. Sallam, R. K. Chakrabortty, and M. J. Ryan, "A two-stage multi-operator differential evolution algorithm for solving resource constrained project scheduling problems," *Future Generation Computer Systems*, vol. 108, pp. 432–444, 2020.

[19] F. Berthaut, R. Pellerin, A. Hajji, and N. Perrier, "A path relinking-based scatter search for the resource-constrained project scheduling problem," *International Journal of Project Organisation and Management*, vol. 10, no. 1, pp. 1–36, 2018.

[20] 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.

[21] G. Koulinas, L. Kotsikas, and K. Anagnostopoulos, "A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem," *Information Sciences*, vol. 277, pp. 680–693, 2014.

[22] R. Zamani, "A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 229, no. 2, pp. 552–559, 2013.

[23] S. Proon and M. Jin, "A genetic algorithm with neighborhood search for the resource-constrained project scheduling problem," *Naval Research Logistics (NRL)*, vol. 58, no. 2, pp. 73–82, 2011.

[24] D. Debels, B. De Reyck, R. Leus, and M. Vanhoucke, "A hybrid scatter search/electromagnetism meta-heuristic for project scheduling," *European Journal of Operational Research*, vol. 169, no. 2, pp. 638–653, 2006.

[25] V. Valls, F. Ballestin, and S. Quintanilla, "Justification and rcpsp: A technique that pays," *European Journal of Operational Research*, vol. 165, no. 2, pp. 375–386, 2005.

[26] S. Hartmann, "A self-adapting genetic algorithm for project scheduling under resource constraints," *Naval Research Logistics (NRL)*, vol. 49, no. 5, pp. 433–448, 2002.