Hindawi
Advances in Operations Research
Volume 2017, Article ID 7048042, 10 pages
https://doi.org/10.1155/2017/7048042

Research Article

Towards Merging Binary Integer Programming Techniques with Genetic Algorithms

Reza Zamani

School of Computing and Information Technology, Wollongong University, Wollongong, NSW 2522, Australia

Correspondence should be addressed to Reza Zamani; reza@uow.edu.au

Received 10 June 2017; Revised 27 August 2017; Accepted 6 September 2017; Published 17 October 2017

Academic Editor: Demetrio Laganà

This paper presents a framework based on merging a binary integer programming technique with a genetic algorithm. The framework uses both lower and upper bounds to make the employed mathematical formulation of a problem as tight as possible. For problems whose optimal solutions cannot be obtained, precision is traded with speed through substituting the integrality constrains in a binary integer program with a penalty. In this way, instead of constraining a variable u with binary restriction, u is considered as real number between 0 and 1, with the penalty of $Mu(1-u)$, in which M is a large number. Values not near to the boundary extremes of 0 and 1 make the component of $Mu(1-u)$ large and are expected to be avoided implicitly. The nonbinary values are then converted to priorities, and a genetic algorithm can use these priorities to fill its initial pool for producing feasible solutions. The presented framework can be applied to many combinatorial optimization problems. Here, a procedure based on this framework has been applied to a scheduling problem, and the results of computational experiments have been discussed, emphasizing the knowledge generated and inefficiencies to be circumvented with this framework in future.

1. Introduction

Combinatorial optimization problems are widespread. They include important instances like network design, job shop scheduling, flow shop scheduling, quadratic assignment, protein alignment, resource-constrained project scheduling, vehicle routing, and many other problems.

In general, the solution strategies used to solve these problems are mainly classified into exact and heuristic methods. Whereas the well-known examples of exact methods are binary integer programming techniques, effective instances of heuristics are genetic algorithms.

One of the endeavours in integrating binary integer programming with genetic algorithm is the one reported in [1], and another work, with respect to this integration, has been reported in [2]. Moreover, there are several other general works related to integrating integer, and not necessarily binary integer, programming with genetic algorithm including those presented in [3–8].

Emphasizing that binary integer programs are very difficult to solve, and their execution times grow exponentially, in [1], a genetic-binary combinatorial algorithm has been presented with a method called "change-zero and add-one." In [2], the combination of a binary integer program with a genetic algorithm has been used for the monitoring, controlling, and protecting of power systems.

In this paper, we present a general framework based on the integration of a binary integer programming technique and a genetic algorithm. The framework converts a binary integer programming problem to a quadratic programming formulation and then by using a genetic algorithm revises the solution obtained by the quadratic programming technique.

The presented framework is called QGA in which Q and GA stand for quadratic programming and genetic algorithm, respectively. In the QGA, the employed genetic algorithm fine-tunes an infeasible solution produced by a quadratic programming technique, which itself solves a relaxed binary integer programming problem. In the corresponding binary integer programming problem, the integrality constrains of binary variables are relaxed, in the sense that they are allowed to be in the range between 0 and 1.

When the integrality constraint is relaxed and a binary variable u is considered as a real number between 0 and 1, a penalty of $Mu(1-u)$, in which M is a large number, is added

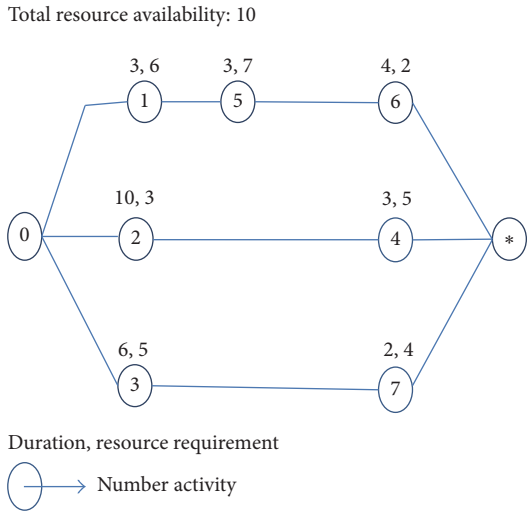

Figure 1: A sample resource-constrained project.

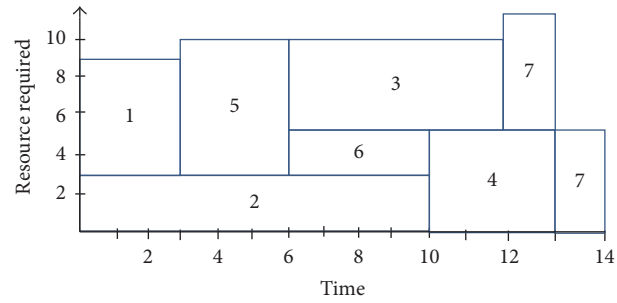

Figure 2: The optimal schedule of the sample project.

to the minimization objective function. These penalties are aimed at preventing those values not near to 0 or 1 from being selected, as $Mu(1-u)$ becomes large for those values of u which are not near to 0 or 1. The produced nonbinary values are then fine-tuned through a genetic algorithm and is converted to a feasible solution.

The QGA is also equipped with a tree search aimed at finding a lower and upper bound for tightening the binary integer programming formulation of the problem. In the cases where the computed lower and upper bounds are equal, the optimal solution has already been obtained and no binary integer programming formulation is required to be generated.

The QGA is a general algorithm which can be applied to many combinatorial optimization problem. Because architecting effective software applications for project scheduling is of prime importance [9], upon the development of the QGA, as a general solution strategy, the Resource-Constrained Project Scheduling Problem (RCPSP) has been applied. As an NP-hard problem, the RCPSP is simply defined as minimizing the duration of a project which includes several activities each requiring a number of scarce resources. All resources are renewable and, during the execution of the project, the availability of each resource is constant.

Each activity has a set of predecessors as well as certain duration and can start only when all of its predecessors have been completed. Preemption is not allowed in the sense that when an activity starts, it should be completed without any interruption. Two starting and ending activities which require no resources and have zero duration represent the starting and ending of the project, respectively. Figure 1 shows a sample resource-constrained project and Figure 2 depicts its optimal schedule.

Unlike heuristics, integer programming, as a systematic search technique, cannot be applied to real-sized problems. Heuristics provided for the RCPSP are categorically diverse. Priority rules comprise a large portions of early heuristic strategies for the RCPSP, and an extensive survey on these methods has been provided in [10]. Priorities are simply converted to a solution through a serial or parallel schedule generation scheme. The first scheme has been presented in [11], and the second one has been developed by Brooks and published in [12].

A survey on serial and parallel schedule generation schemes has been provided in [13]. Also a survey on deterministic project scheduling has been provided in [14] and extended in [15].

Regardless of whether a schedule is constructed by the serial or parallel schedule generation scheme, the schedule can be double justified for possible improvement [16]. By double justification, the authors mean the left and right justification. The concept of the left and right justified schedules dates back to sixties and was used towards facilitating the computation of slack values of activities in the presence of resource constraints [17].

Based on right and left justification of activities, the iterative forward/backward scheduling technique presented in [18] iteratively applies serial forward and backward scheduling for several times. In effect, double justification mentioned is an improvement over this iterative forward/backward scheduling technique. Also nearly the same as double justification, in [19], a technique has been presented which, for improving a schedule, uses a backward scheduling followed by a forward schedule. In [15], the notion of "forward-backward improvement" has been used to refer to both approaches presented in [16, 19].

Point-based and population-based algorithms comprise two main categories of heuristics, referred to as local searches and evolutionary algorithms, respectively. A key point with the efficiency of these two categories of algorithms in solving the RCPSP is the concept of encoding or representation. In this regard, the standardized random key representation presented in [20] is of prime importance. This representation, which is an improvement over the representation presented in [21], assigns the same order to activities which have started at the same time.

Heuristics and systematic searches, despite being traditionally used in two different contexts, can cooperate to provide better solution strategies. In this cooperation, a heuristic can use information generated in the middle of a systematic search to steer the search more effectively.

The rest of the paper is as follows. Section 2 describes the related work with respect to the RCPSP, and Section 3 discusses the QGA and its application to the RCPSP. Computational experiments are presented in Section 4. Section 5 provides concluding remarks and future directions for improving the framework.

2. Related Work

The related work is presented in two subsections. The first subsection discusses the related work to the integer linear programming formulation of the RCPSP, and the second subsection reviews the corresponding heuristics.

2.1. Integer Linear Programming Formulation of the RCPSP. The basic discrete time formulation provided for the RCPSP was presented in [22] and its disaggregated version was late provided in [23]. These two formulations have been compared in [24], and their difference has been highlighted. The results indicate that formulating precedence constraints makes the LP relaxation of the second formulation tighter than that of the first one.

Based on a linear programming formulation, a procedure has been presented in [25] which can be considered as an improvement over the procedure developed in [26]. This improved procedure tightens the initial formulation of the linear programs by an effective constraint propagation mechanism and consequently solves the linear programming formulation obtained. In this procedure, an RCPSP instance is described through a set of starting times for activities, with the constraints being actively propagated to reduce the combinatorial effort needed to find the starting times.

As another effective method, we can name the linear programming formulation provided in [27] which has been further strengthened in [28] by additionally considering a time window for each activity. Defined based on a hypothetical upper bound as well as the heads and tails of activities, these time windows make formulation highly effective. In this method, a constraint propagation mechanism has been used to facilitate the search process.

In [29], constraint programming is performed as a preprocessing step for a linear program that relaxes integrality constraints, adding valid linear inequalities to the resulting relaxation towards strengthening the bound.

The flow-based continuous formulations presented in [30–32], as three effective formulations for the RCPSP, have been compared with one another in [24]. In the first formulation, variables include (i) the starting time variables, which show the starting time of activates, (ii) sequential binary variables indicating whether an activity is processed after another activity, and (iii) flow variables indicating the amount of a particular resource transferred from one activity to another.

On the other hand, in the second formulation, which is based on rectangle packing, an effective insertion technique is used. Unlike the first two formulations, the third formulation is limited to handling projects requiring no more than 2 scarce resources.

After making these comparisons, the authors in [24] have presented two linear programming formulations which work based on the concept of event, namely, *Start/End* and *On/Off*. Unlike both discrete time and flow-based formulations, in event-based formulations, the emphasis is on indexing variables based on events. The number of events cannot exceed the number of activities plus one. In their *Start/End* formulation, the left-shift rule is exploited in the sense that the starting time of an activity beginning after time zero should coincide with the finishing time of another activity.

In their *Start/End* formulation, two types of binary variables have been used representing whether each activity starts and ends at an event, respectively, assuming that each activity starts at an event and ends at another event. Resource conservation constraints enforce activities which end at an event to release the resources they have taken and activities which start at an event to seize their required resources.

In their *On/Off* formulations, on the other hand, only one type of binary variables has been used, representing whether, in an event, an activity starts or is still in-process. As well as these single-type binary variables, a single continuous variable is used to show the end of the project. The same within the *Start/End* formulation, in this formulation, the starting time of each event is shown with a continuous variable.

An unintentionally omitted part of the aforementioned *Start/End* formulation has been identified in [33] and then it has been added to the formulation. Without this part, there could be chances for an activity to end without even being started.

In comparison with formulations indexed by time or flow, event-based formulations have the advantage of being involved with a smaller number of variables. That is why these formulations work well for projects whose scheduling horizon is comparatively long [24]. Interestingly, the authors, based on their computational experiments, emphasized that discrete time formulation is still one of the best formulations possible and that they work better than other formulations on many instances. That is why in our paper, the discrete time formulation has been adopted.

2.2. Corresponding Heuristics. In this subsection, we classify heuristics for the RCPSP into three classes of point-based heuristics, population-based heuristics, and others. Point-based heuristics simply maintain and manipulate a single solution in each stage, whereas population-based heuristics are based on maintaining and improving multiple candidate solutions.

2.2.1. Point-Based Heuristics. Two Tabu search procedures have been presented in [34]. The first procedure has a simple neighbourhood structure aimed at eliminating critical paths. On the other hand, the neighbourhood structure in the second procedure is aimed at placing activities in parallel or deleting parallelity relations. The second procedure is based on the fact that each pair of activities can be in disjunction, conjunction, parallel, or flexible relation and a heuristic can convert new set of relations to a schedule.

Sampling methods employ a schedule generation scheme and a priority rule and then generate different schedules by biasing the priority rule and converting the biased rule to a schedule. In [35], based on extensive computational experiments, it has been indicated that the best metaheuristics perform consistently better than the best sampling approaches.

The bidirectional method presented in [36] and the hybrid multipass method presented in [19] can be considered as two effective point-based methods. The first procedure simultaneously employs forward and backward partial schedules and, in each iteration, among all eligible activities in the forward and backward directions, selects an activity based on a priority rule.

In the case where an activity can be scheduled in both directions, some other rules are employed to determine the direction in which the activity should be scheduled. Both partially constructed forward and backward schedules join to create a complete schedule. On the other hand, in the second procedure, a biased random sampling component is combined with forward-backward passes. The procedure performs iteratively forward and backward passes and is terminated as soon as no improvement is achieved.

An ant colony optimization procedure has been presented in [37] which employs artificial ants to search for good solutions through making step-by-step probabilistic decisions. These decisions, which are related to find the order of activities, are made based on the latest finish times of activities as well as positions in the activity list which have been found promising by other ants for placing each particular activity.

Using a simulated annealing method, the procedure presented in [38] employs an effective neighbourhood in which for finding the neighbours of an activity list first an activity is randomly selected and the positions of its latest predecessor and earliest successor are found. Then the selected activity can move to all the places between these two extreme points.

A large neighbourhood search presented in [39] fixes some parts of a decision vector and finds suitable values for the remaining variables as a subproblem. This can be performed through either an exact method or a heuristic. Since the method can be performed for several rounds, the information obtained in previous rounds can be used in a new round to improve the results.

A forceful move strategy, called enhanced move, has been presented in [40] that shifts an activity forward (backward) in any range along with its successors (predecessors), extending the left (right) limit of a boundary that an activity can shift.

The insertion technique presented in [41] deletes a number of activities from the activity list and inserts them in the activity list based on an inserting module, causing them to have a new and possibly more effective order. The inserting module searches all feasible positions to find the best insertion point for each activity to insert.

An annealing-like search procedure has been presented in [42] which, by cooling an initial environment, based on a decreasing temperature, converts the order of activities to a schedule. In general, the better the quality of a solution, the higher its chance of acceptance. In effect, when the temperature is very low, only improving solutions can replace the modified solution whereas when the temperature is high, the chance of selecting nonimproving solutions is not small. The authors have also used an effective jumping and walking mechanisms to improve the performance of their procedure.

2.2.2. Population-Based Heuristics. A self-adapting genetic algorithm has been presented in [43] in which the decoding mechanism is determined through a gene in the sense that depending on the value of this gene, the serial or parallel schedule generation scheme is used for the decoding purposes. In [21] a two-phase method has been presented whose first phase consists of a combination of path relinking and scatter search. Whereas the path relinking component explores trajectories which connect elite solutions, the scatter search component employs the linear combinations of subsets of solution vectors, producing new schedules which inherit from the high quality schedules. This is done in a high quality population through using the spanned convex region of its schedules. The vicinity of the solution produced in the first phase is searched in depth to further improve its results in the second phase.

In the schedules generated by the genetic algorithm presented in [44], no resource is kept idle for more than a predefined period if such resource could start at least one activity. In other words, in this genetic algorithm, the corresponding schedules vary between nondelay and active schedules. On the other hand, through generating both nondelay and active schedules, the population-based method presented in [45] uses a population of size 2 and iteratively replaces the individual with the worst quality with new individuals generated.

The genetic algorithm presented in [46] uses a multiagent model in which agents are placed in a lattice-like environment and through communicating with their neighbours improve their performance. In this lattice-like environment, each agent has four neighbours which can be changed in the search process.

Rather than using a lattice-like environment, the genetic algorithm used in [47] uses a simple pool with biased random keys as its representation and employs forward-backward concept as well as chromosome adjustment to reflect the improvement made in the solutions generated.

In general, crossover operators play a key role in the efficiency of population-based procedures, and hence the efforts made in [48, 49] in the development of peak crossover and magnet-based crossover, respectively, are worth mentioning.

The particle swarm optimization technique presented [50] adaptively eliminates invalidity of generated positions and guarantees the feasibility of these new positions. An operator called valid particle generator, which uses the in- and out-degree of activities shown in activity networks, performs this elimination. An adaptive inertia weight determines the effect of previous speed on the new speed.

2.2.3. Other Heuristics. Among other heuristics presented for the problem are decomposition techniques. These heuristics decompose a project into a number of subprojects, and, after scheduling these subprojects separately, concatenate the results to find a solution for the original project. The

decomposition procedures developed for the RCPSP are those presented in [51–54].

The hybrids presented in [55–60] can also be classified in this subsection.

Whereas the first hybrid combines neural networks with genetic algorithms, the second hybrid combines greedy selection with simulated annealing and Tabu search, and the third algorithm relies on forward-backward improvement in the context of scatter search and bidirectional path relinking.

The fourth hybrid is a hyperheuristic operating based on particle swarm optimization. By the term hyperheuristic, their authors mean an upper level heuristic which controls several other heuristics. The fifth hybrid is based on the concept of A-Team, which is a set of agents with a shared memory for keeping the best obtained solution obtained. The employed agents in this software environment use a combination of a simple local search, Tabu search, path relinking, and crossover-based techniques to produce solutions. In each iteration, with the chance of p, the worst solution in the team is replaced with a fresh random solution. The sixth hybrid employs a simulated annealing technique which fine-tunes the results obtained in a genetic algorithm using a pool of solutions.

3. The QGA

Using both lower and upper bounds to make the employed mathematical formulation of the problem as tight as possible, the QGA seeks to find the optimal solutions of problems through its tree search component. For problems whose optimal solutions cannot be guaranteed, the QGA can effectively trade precision with speed through substituting the integrality constrains in the corresponding binary integer linear programming model with penalties.

With removing the integrality constraints, the original binary variables can be set to any real number, between zero and one, and this makes the solution infeasible. The employed genetic algorithm makes this infeasible solution feasible with keeping makespan as low as possible. The quadratic programming problem generated and its solution strategy are aimed at keeping the relaxed binary variables near to 0 or 1 as much as possible, preventing the even distribution of these relaxed variables in their range of 0-1. Figure 3 depicts the modules of the QGA and the order in which they are employed.

As is seen the tree search module, which is aimed at finding an initial lower and upper bound for the problem is followed by an integer program, which tests whether or not the problem can be solved. In the case the optimal solution cannot be guaranteed within a specified time, a quadratic programming problem is generated through substituting the integrality constraints with a quadratic objective function. This is done as follows.

Assume u is a variable which should be either 0 or 1. Instead of constraining u with binary restriction, we can indicate that u is real number between 0 and 1 and then add $Mu(1-u)$, in which M is a large number, to our minimization objective function. Note that values which are not near 0 and 1 make the component of $Mu(1-u)$ large and are expected to be avoided implicitly.

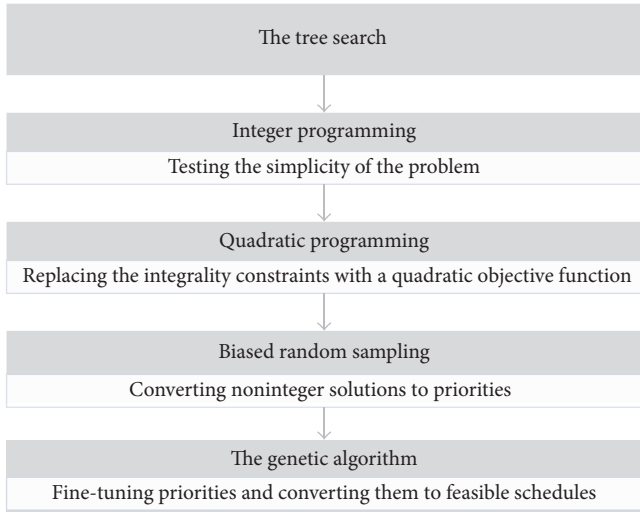

Figure 3: The constituent modules of the QGA.

As is seen in Figure 3, upon generating a solution by the quadratic programming, the braised random sampling is used to convert the solution in which each activity has different starting times with different weights to a feasible solution. In the last stage, the genetic algorithm uses these priorities to fill its initial pool and find a feasible schedule. Algorithm 1 presents the pseudocode of the QGA.

As is seen in line (4) of the pseudocode, if the tree search obtains a solution, the QGA is terminated.

Lines (7) through (14) of the pseudocode are performed based on the upper and lower bounds produced. Whereas finding the lower bound is involved with traversing the search tree in line (2), the upper bound is found heuristically.

The objective function and constraints are constructed in lines (7), (8), respectively. In line (9), the binary integer program is constructed. Line (10) converts the integer programming model to the quadratic programming model through removing all integrality constraints and adding the penalty of $Mu(1-u)$ to the objective function for each original binary variable u, with M being a large number. Line (13) converts the solution obtained by the quadratic programming technique to the priorities guiding the genetic algorithm. Finally, as line (14) indicates, these priorities are used by the employed genetic algorithm to produce a feasible solution for the problem.

As line (2) of the pseudocode indicates, the first operation performed by the QGA is the tree search. With respect to applying the QGA to the RCPSP, the employed tree search has been selected as the procedure presented in [61].

The employed tree search starts by putting the null schedule at the root of tree and expanding partial schedules with minimum lower bound one after another. With expanding a partial schedule, new partial schedules are added to the tree as its leaf nodes, and the lower bound associated with each new partial schedule is calculated. The search stops when either a complete schedule is obtained or the time limit prevents it

```
(1)  PROCEDURE QGA()
(2)    Do tree search and find a lower bound.
(3)    IF (Tree search has found a solution)
(4)          Print the solution as an optimal solution.
(5)    ELSE
(6)        Do Heuristic Search and find an upper bound.
(7)        Construct the objective function of a binary integer program.
(8)        Construct the constraints of the binary integer program.
(9)        Assemble the binary integer program.
(10)       In the integer program, replace all integerality constraints with the corresponding penalty.
(11)       Modify the objective function for preventing even distribution of variables in the range 0-1.
(12)       Solve the quadratic program (linear program with quadratic objective function).
(13)       Convert the solution obtained to the priorities directing the genetic algorithm.
(14)       Print the obtained solution by the genetic algorithm.
(15)   ENDIF
(16)   ENDPROCEDURE
```

Algorithm 1: The pseudocode of the QGA.

```
(1)   PROCEDURE CreateObjectiveFunctions //Creating F1, F2, F3, F4, F5 as fiveobjective Functions
(2)       //F1: Makespan
(3)       //F2: AvgStartTimeOfActivities criterion subject to Makespan = lower bound
(4)       //F3: AvgStartTimeofActivitiesWeightedByLongestPathRatio subject to Makespan = lower bound
(5)       //F4: μF1 + (1 − μ)F2 without being subject to setting Makespan to lower bound
(6)       //F5: μF1 + (1 − μ)F3 without being subject to setting Makespan to lower bound
(7)       SET TotalLongestPath to 0.
(8)       FOR each activity i
(9)           Add TotalLongestPath by the longest path of activity i.
(10)      ENDFOR
(11)      SET k to 1. //k is index of semi binary variables.
(12)      FOR each activity i
(13)          FOR each t between the earliest and the latest start ofactivity i
(14)              IF (activity i is the ending activity of project)
(15)                  SET F1[k] to t.
(16)              ELSE
(17)                  SET F1[k] to 0.
(18)              ENDIF
(19)              SET F2[k] to 1 divided by the number of activities.
(20)              SET F3[k] to thelongest path of activity i divided by TotalLongestPath.
(21)              SET F4[k] to μF1[k] + (1−μ)F2[k].
(22)              SET F5[k] to μF1[k] + (1−μ)F3[k].
(23)              Add k by 1;
(24)          ENDFOR
(25)      ENDFOR
(26)  ENDPROCEDURE
```

Algorithm 2: The pseudocode of the CreateObjectiveFunctions component of the QGA.

from expanding any partial schedule. Whereas, in the first case, it guarantees the optimality of the complete schedule constructed, in the second case, it provides a lower bound for the problem.

Upon the completion of the tree search and if the tree search has not guaranteed the optimality of the obtained schedule, the QGA, as is shown in lines (6)–(9) of the pseudocode presented in Algorithm 1, first computes an upper bound and then, based on this upper bound, constructs the objective function and constraints of the integer programming model. Substituting all integrality constraints with penalties, it then modifies the objective function to prevent the even distribution of variables in the range 0-1. Algorithm 2 presents the pseudocode of the module performing such modification.

As is seen in Algorithm 2, five different criteria are used, namely, $F1$ through $F5$. The first criterion, $F1$, is the original makespan criterion. The second criterion, $F2$, is the average

starting times of activities, and associated with this criterion there is a constraint that forces makespan to be the same as the lower bound. Note that the removal of integrality constraints highly has relaxed the entire constraints and adding this new constraint cannot overtight them.

The third criterion, $F3$, is the weighted version of $F2$, in which the starting time of each activity is weighted by its corresponding longest path. This is aimed at forcing activities with larger longest path values to be started sooner.

The fourth criterion, $F4$, is a linear combination of the first and third criteria. As is seen in the pseudocode, the value of μ, which is between 0 and 1, determines the similarity of this criterion with each of its two components. The same is with the fifth criterion, $F5$, which is a linear combination of the first and fourth criteria.

As is shown in line (13) of the pseudocode presented in Algorithm 1, after modifying the objective function and solving the quadratic programming model, the QGA converts the obtained solution to the priorities guiding the genetic algorithm. Having completed the description of the QGA, the following discussion only applies to the application of the QGA to the RCPSP.

As a result of solving the quadratic programming model, a vector X is obtained which can be portioned into n portions, with each portion allocated to one of the activities. Assuming that the earliest start and latest finish times of activity i are represented with ESi and LSi, respectively, then the first $LS_1 - ES_1 + 1$ elements of X are associated with activity 1, its next $LS_2 - ES_2 + 1$ elements are associated with activity 2, and so forth.

Note that, in X, the sum of elements associated with each activity is one, because, after all, an activity needs to be started between its earliest and latest start times, and removing integrality constraints eliminates the binary nature of starting time variables. The employed biased random sampling guarantees that the higher the value of a starting time for an activity, the higher the chance that the activity receives the corresponding starting time as its priory.

In the employed genetic algorithm, we have used the decoding discussed in [20]. Moreover, in line with [43], a two-point crossover operator has been employed and the genetic algorithm uses either serial or parallel schedule generation scheme as its decoding mechanism. In the same fashion stated in [43], a gene determines whether the serial or parallel schedule generation scheme should be used, and another gene determines whether the forward or backward scheduling should be used.

Regardless of whether a serial or parallel schedule generation scheme is employed for the decoding purposes, a triple-justification mechanism has been applied to possibly enhance the result. By triple-justification, we mean a modification made on double justification [16]. For a schedule generated in a forward manner, in this modified structure, a backward scheduling is followed by a forward one which is followed by a second backward scheduling. For the schedules generated in the backward manner, the directions are reverse. In other words, for these cases, a forward scheduling is followed by a backward one, which is followed by a second forward scheduling.

4. Computational Results

A PC under Windows operating system with 2.5 GHZ speed and 8 MB RAM has been used to test the performance of the QGA on 600 benchmark instances with 120 activities in [62]. These instances are available in the PSPLIB, and in all of them the duration of activities has been drawn from the interval [1, 10], with each instance including 4 types of resources.

As Figure 3 shows, the QGA have 5 components. MATLAB has been used for coding the quadratic programming component, and C++ has been used to program other four components.

First, with all of its five components, we tested the QGA on a small number of benchmark instances to find the speed of these components as well as of the effect of different parameters on their performance. Since the computer programs for these components are currently separate from one another, testing is a time-consuming task.

The experiments showed that whereas the combination of the first and last components, as boundary modules, works comparatively fast, adding the three middle components causes the QGA to require orders of magnitude more time than that required by those two boundary modules.

In effect, for nearly all tests performed, the quadric programming component could not solve an instance within 20 minutes. It seems this is mainly because of the concave components, in the form of $Mu(1 - u)$, in the objective function of the quadratic program.

One promising strategy to circumvent this difficulty could be to set a time limit and asking MATLAB to stop its quadratic programming process and send its intermediate output as soon as this time limit is reached. After all, we are not looking for an exact or even feasible solution; as such a solution is fine-tuned by the genetic algorithm module. Because, at this stage, the modules are separate from one another; this strategy, despite all of its merits, was not used, and. for practicality purposes, we simply bypassed the three middle components, combining only components 1 and 5, in a single procedure. Note that in the cases where the first component can find an optimal solution, the fifth component is not called at all. Moreover, the fifth component has to create its initial genomes randomly without using the results obtained in the first or middle components.

The population size and the number of generations in the fifth component were set to 200 and 63, leading to the generation of 50400, $200 * 4 * 63$, schedules. The number 4 in this calculation has appeared because triple-justification increases the number of each schedule from 1 to 4, $1 + 3$. We also assigned 0.5 seconds to the first component. In this way, it takes, on average, 5 seconds for each of 600 instances to be solved. For this setting, the average percentage deviation from the CPM (Critical Path Method) lower bound is 30.76%.

The results produced for these 600 instances have also been compared with the best available solutions in the PSPLIB. This comparison shows that, for 297 of these instances, the produced solution is equal to the best available solution in the literature. Moreover, on average, the percentage deviation from the best available solutions for other 303 instances is 1.9%.

5. Conclusion

The binary integer programming technique and the genetic algorithm are two distinctly different solution strategies for solving combinatorial optimization problems. In this regard, the keystone of the QGA is its ability of combining binary integer programming with a genetic algorithm.

The QGA uses quadratic programming, which is the process of solving a special type of mathematical optimization problem with linear constraints but quadratic objective function. It is aimed at managing an exploration-exploitation trade-off by activating its quadratic programming component for identifying a high quality area, which is thoroughly searched by its genetic algorithm component.

The QGA can be applied to many combinatorial optimization problem, like the flow shop [63] and job shop scheduling [64], but in this paper, it was solely applied to the RCPSP.

Its application to the RCPSP showed that, among its five components, the combination of the first and last components works comparatively fast. It also showed that the main problem with the QGA is that $Mu(1-u)$ is a concave function of u, and this increases the execution time of its quadratic programming component.

This means that the genetic algorithm cannot use the solutions produced by the middle components to guide its search and has to create its initial pool randomly. Two solutions to this issue may be envisaged.

The first solution is to set a time limit and force the quadratic program to stop and output the intermediate solution upon reaching that time limit. After all, the solutions produced by the quadratic program need not be exact, or even feasible, as it is the responsibility of the genetic algorithm to make these solutions feasible. The second solution is to find penalties that are easier to handle.

The performance of the framework can also be improved by augmenting a module which can learn the best values of the parameters. This learning process can occur based on the characteristics of the problem at hand. For instance, a new genetic algorithm can be employed in which different genes represent different parameters. In such a modified algorithm, the solution obtained through the corresponding parameters determines the fitness of parameters.

In many circumstances, binary integer programs are extremely time-consuming. On the other hand, genetic algorithms, if guided correctly, can produce high quality solutions in short time. Integrating binary integer programming techniques with genetic algorithms, as two distinct but powerful solution strategies, is of prime importance and requires considerable attention.

Conflicts of Interest

The author declares that there are no conflicts of interest.

References

[1] Y. Yan, "Genetic-binary combinatorial algorithm for 0-1 integer programming," in *Proceedings of the International Workshop on Autonomous Decentralized System*, pp. 205–209, September 2000.

[2] N. P. Theodorakatos, N. M. Manousakis, and G. N. Korres, "Optimal placement of PMUs in power systems using binary integer programming and genetic algorithm," in *Proceedings of the 9th Mediterranean Conference on Power Generation, Transmission Distribution and Energy Conversion, MedPower 2014*, November 2014.

[3] J. A. Joines, C. T. Culbreth, and R. E. King, "Manufacturing cell design: an integer programming model employing genetic algorithms," *IIE Transactions*, vol. 28, no. 1, pp. 69–85, 1996.

[4] T. Yokota, M. Gen, and Y.-X. Li, "Genetic algorithm for non-linear mixed integer programming problems and its applications," *Computers & Industrial Engineering*, vol. 30, no. 4, pp. 905–917, 1996.

[5] Y.-C. Lin, K.-S. Hwang, and F.-S. Wang, "A mixed-coding scheme of evolutionary algorithms to solve mixed-integer nonlinear programming problems," *Computers & Mathematics with Applications. An International Journal*, vol. 47, no. 8-9, pp. 1295–1307, 2004.

[6] K. Deep, K. P. Singh, M. L. Kansal, and C. Mohan, "A real coded genetic algorithm for solving integer and mixed integer optimization problems," *Applied Mathematics and Computation*, vol. 212, no. 2, pp. 505–518, 2009.

[7] C. F. M. Toledo, L. De Oliveira, R. De Freitas Pereira, P. M. França, and R. Morabito, "A genetic algorithm/mathematical programming approach to solve a two-level soft drink production problem," *Computers & Operations Research*, vol. 48, pp. 40–52, 2014.

[8] L. Poli, G. Oliveri, and A. Massa, "An integer genetic algorithm for optimal clustering in phased array antenna," in *Proceedings of the 2017 International Applied Computational Electromagnetics Society Symposium - Italy (ACES)*, pp. 1-2, Florence, March 2017.

[9] R. Zamani, R. B. Brown, G. Beydoun, and W. J. Tibben, "The Architecture of an Effective Software Application for Managing Enterprise Projects," *The Journal of Modern Project Management*, vol. 5, no. 1, 2017.

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

[11] J. Kelley, "The critical-path method: resource planning and scheduling," in *Industrial Scheduling*, J. F. Muth and G. L. Thompson, Eds., pp. 347–365, Prentice-Hall, Upper Saddle River, NJ, USA, 1963.

[12] D. D. Bedworth and J. E. Bailey, *Integrated Production Control Systems-Management, Analysis, Design*, Wiley, New York, NY, USA, 1982.

[13] 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.

[14] R. Kolisch and R. Padman, "An integrated survey of deterministic project scheduling," *Omega* , vol. 29, no. 3, pp. 249–272, 2001.

[15] 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.

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

[17] J. D. Wiest, "Some Properties of Schedules for Large Projects with Limited Resources," *Operations Research*, vol. 12, no. 3, pp. 395–418, 1964.

[18] 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.

[19] P. Tormos and A. Lova, "A competitive heuristic solution technique for resource-constrained project scheduling," *Annals of Operations Research*, vol. 102, pp. 65–81, 2001.

[20] 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.

[21] V. Valls, F. Ballestin, and S. Quintanilla, "A population-based approach to the resource-constrained project scheduling problem," *Annals of Operations Research*, vol. 131, pp. 305–324, 2004.

[22] A. A. B. Pritsker and L. J. Watters, A zero-one programming approach to scheduling with limited resources, RAND Corporation, Santa Monica, Calif, USA, 1968.

[23] N. Christofides, R. Alvarez-Valdes, 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.

[24] O. Koné, C. Artigues, P. Lopez, and M. Mongeau, "Event-based MILP models for resource-constrained project scheduling problems," *Computers & Operations Research*, vol. 38, no. 1, pp. 3–13, 2011.

[25] P. Baptiste and S. Demassey, "Tight LP bounds for resource constrained project scheduling," *OR Spectrum*, vol. 26, no. 2, pp. 251–262, 2004.

[26] 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.

[27] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco, "An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation," *Management Science*, vol. 44, no. 5, pp. 714–729, 1998.

[28] P. Brucker and S. Knust, "Linear programming and constraint propagation-based lower bound for the RCPSP," *European Journal of Operational Research*, vol. 127, no. 2, pp. 355–362, 2000.

[29] S. Demassey, C. Artigues, and P. Michelon, "Constraint-propagation-based cutting planes: an application to the resource-constrained project scheduling problem," *INFORMS Journal on Computing*, vol. 17, no. 1, pp. 52–65, 2005.

[30] R. A.-V. Olaguíbel and J. T. Goerlich, "The project scheduling polyhedron: Dimension, facets and lifting theorems," *European Journal of Operational Research*, vol. 67, no. 2, pp. 204–220, 1993.

[31] C. Artigues, P. Michelon, and S. Reusser, "Insertion techniques for static and dynamic resource-constrained project scheduling," *European Journal of Operational Research*, vol. 149, no. 2, pp. 249–267, 2003.

[32] M. Sabzehparvar and S. M. Seyed-Hosseini, "A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags," *The Journal of Supercomputing*, vol. 44, no. 3, pp. 257–273, 2008.

[33] C. Artigues, P. Brucker, S. Knust, O. Koné, P. Lopez, and M. Mongeau, "A note on "Event-based MILP models for resource-constrained project scheduling problems"," *Computers & Operations Research*, vol. 40, no. 4, pp. 1060–1063, 2013.

[34] T. Baar, P. Brucker, and S. Knust, "Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem," in *Meta-heuristics: advances and trends in local search paradigms for optimization (Sophia-Antipolis, 1997)*, pp. 1–18, Kluwer Acad. Publ., Boston, Mass, USA, 1999.

[35] 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.

[36] R. Klein, "Bidirectional planning: Improving priority rule-based heuristics for scheduling resource-constrained projects," *European Journal of Operational Research*, vol. 127, no. 3, pp. 619–638, 2000.

[37] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 333–346, 2002.

[38] 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.

[39] M. Palpant, C. Artigues, and P. Michelon, "LSSPER: solving the resource-constrained project scheduling problem with large neighbourhood search," *Annals of Operations Research*, vol. 131, no. 1, pp. 237–257, 2004.

[40] K. Fleszar and K. S. Hindi, "Solving the resource-constrained project scheduling problem by a variable neighbourhood search," *European Journal of Operational Research*, vol. 155, no. 2, pp. 402–413, 2004.

[41] I. Pesek and J. Žerovnik, "Best insertion algorithm for resource-constrained project scheduling problem," in *11th International Conference on Operational Research*, Pula, Croatia, 2006.

[42] A. Lim, H. Ma, B. Rodrigues, S. T. Tan, and F. Xiao, "New meta-heuristics for the resource-constrained project scheduling problem," *Flexible Services and Manufacturing Journal*, vol. 25, no. 1-2, pp. 48–73, 2013.

[43] 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.

[44] J. J. Mendes, J. F. Goncalves, and M. G. Resende, "A random key based genetic algorithm for the resource constrained project scheduling problem," *Computers & Operations Research*, vol. 36, no. 1, pp. 92–109, 2009.

[45] R. Zamani, "An accelerating two-layer anchor search with application to the resource-constrained project scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 975–984, 2010.

[46] X. Pan and H. Chen, "A multi-agent social evolutionary algorithm for resource-constrained project scheduling," in *Proceedings of the 2010 International Conference on Computational Intelligence and Security, CIS 2010*, pp. 209–213, chn, December 2010.

[47] J. F. Gonçalves, M. G. C. Resende, and J. J. M. Mendes, "A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem," *Journal of Heuristics*, vol. 17, no. 5, pp. 467–486, 2011.

[48] V. Valls, F. Ballestín, and S. Quintanilla, "A hybrid genetic algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 185, no. 2, pp. 495–508, 2008.

[49] 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.

[50] N. Kumar and D. P. Vidyarthi, "A model for resource-constrained project scheduling using adaptive PSO," *Soft Computing*, vol. 20, no. 4, pp. 1565–1580, 2016.

[51] A. Sprecher, "Network decomposition techniques for resource-constrained project scheduling," *Journal of the Operational Research Society*, vol. 53, no. 4, pp. 405–414, 2002.

[52] R. Zamani, "An efficient time-windowing procedure for scheduling projects under multiple resource constraints," *OR Spectrum*, vol. 26, no. 3, pp. 423–440, 2004.

[53] 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.

[54] R. Zamani, "A hybrid decomposition procedure for scheduling projects under multiple resource constraints," *Operational Research*, vol. 11, no. 1, pp. 93–111, 2011.

[55] A. Agarwal, S. Colak, and S. Erenguc, "A neurogenetic approach for the resource-constrained project scheduling problem," *Computers & Operations Research*, vol. 38, no. 1, pp. 44–50, 2011.

[56] P. P. Das and S. Acharyya, "Hybrid local search methods in solving resource constrained project scheduling problem," *Journal of Computers (Finland)*, vol. 8, no. 5, pp. 1157–1166, 2013.

[57] F. Berthaut, R. A. Pellerin, N. Hajji, and Perrier., *A Path Relinking-Based Scatter Search for the Resource-Constrained Project Scheduling Problem*, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Canada, 2014.

[58] 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.

[59] P. Jedrzejowicz and E. Ratajczak-Ropel, "Reinforcement Learning strategies for A-Team solving the Resource-Constrained Project Scheduling Problem," *Neurocomputing*, vol. 146, pp. 301–307, 2014.

[60] 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, Article ID 04014082, 2015.

[61] R. Zamani, "An evolutionary implicit enumeration procedure for solving the resource-constrained project scheduling problem," *International Transactions in Operational Research*, vol. 24, no. 6, pp. 1525–1547, 2017.

[62] R. Kolisch and A. Sprecher, "PSPLIB—a project scheduling problem library," *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997.

[63] M. Amirghasemi and R. Zamani, "An effective evolutionary hybrid for solving the permutation flowshop scheduling problem," *Evolutionary Computation*, vol. 25, no. 1, pp. 87–111, 2017.

[64] M. Amirghasemi and R. Zamani, "An effective asexual genetic algorithm for solving the job shop scheduling problem," *Computers & Industrial Engineering*, vol. 83, article no. 3956, pp. 123–138, 2015.