RAIRO-Oper. Res. 50 (2016) 91–118
DOI: 10.1051/ro/2015014

RAIRO Operations Research
www.rairo-ro.org

A MODIFIED MODELING APPROACH AND A HEURISTIC PROCEDURE FOR THE MULTI-MODE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM WITH ACTIVITY SPLITTING

Fatemeh Faghih-Mohammadi[1], Abbas Seifi[1] and Mohammad Khalighi-Sikaroudi[1]

Abstract. This paper presents a new heuristic method for solving multi-mode resource constrained project scheduling problems with renewable resources. Assumptions such as resource vacation and activity splitting are also considered. The proposed heuristic determines one mode for the execution of each activity first, so that the multi-mode problem is reduced to a single-mode one. Our method is compared to two of the existing methods in terms of computational time and solution quality. The results show that while it can outperform one of them (especially as the size of the problem grows), it is outperformed by the other for tested instances with low complexity, but can yield good results for tested instances with higher values of this parameters. This quality may be useful in real-world scheduling problems. We also validate the first phase of our method, *i.e.*, mode selection, with numerical experiments. Our results indicate that better mode vectors selected in the first phase lead to better makespans for the MRCPSP. Moreover, we correct some erroneous constraints in the mathematical model of the problem. We implement the mathematical programming model of the problem in GAMS, and show that solving it to optimality requires much more computational effort compared to our method.

Mathematics Subject Classification. 90B35, 68M20, 91B32.

Received April 23, 2014. Accepted April 17, 2015.

1. Introduction

The multi-mode resource constrained project-scheduling problem (MRCPSP) aims to assign some resources to a set of jobs or activities that have some precedence relations. A common objective of the MRCPSP is minimizing the project makespan while satisfying the precedence relations among the activities. Also, there are always resource constraints that have to be satisfied. Each activity can be performed in one out of several different modes *i.e.*, the activity duration in each mode would depend on the type and amount of resources needed. For example, one skillful worker could finish an activity in 3 days (mode 1), whereas two normal workers might finish the same activity in 2 days (mode 2).

There are three types of resources: Renewable, non-renewable and doubly constrained. Renewable resources have a limited amount available for each period such as manpower and machine hours. Non-renewable resources

Keywords. Project scheduling, resource constrained, multi-mode, splitting, renewable resources.

[1] Department of Industrial Engineering, Amirkabir University of Technology, Hafiz St. 424, P.O. Box 15875-4413, Tehran, Iran.
torino_fm@yahoo.com; aseifi@aut.ac.ir; mks_63@aut.ac.ir

Article published by EDP Sciences

have a fixed total capacity like the budget of a project. Doubly constrained resources have both types of capacity limitations and can be covered by the two other resource categories [72] Real examples of MRCPSPs are scheduling of engineering design activities in product development, scheduling of activities in financial audits and assignment of coding activities in software development projects [21]

Activity splitting may significantly reduce a project makespan especially when resource availability levels are low [21]. In this paper, we describe how activities can be split to achieve significant improvements on the resource consumption level and the makespan of the project.

In some real world situations we may encounter such problems where resources are unavailable in some periods during project execution. These times are known in advance. This unavailability could be due to the absence of some staff and maintenance or overhauls of the equipment in some periods. This kind of resource unavailability is called resource vacation herein [21]

In recent years the growing number of published ISI articles shows that more attention has been paid to scheduling. There are also many extensions to the MRCPSP, such as minimal and maximal time lags between activities (MRCPSP/Max), mode-dependent time lags, splitting and resource dependant processing times. For example, Heilmann [37] presented a heuristic procedure for the MRCPSP with temporal constraints (time lags between activities). His method is evaluated for problem instances with 100 activities and up to 5 modes per activity. Hartmann [35] presented a genetic algorithm along with a local search. In reference [38] the ability of the branch-and-bound algorithm in solving the MRCPSP is evaluated. In reference [48], Parallel and serial scheduling generation schemes (SGS) are given for the MRCPSP. Serial SGS's move forward activity by activity until all activities are scheduled while parallel ones move from one scheduled time period to the next [21]. In references [15, 64] it was shown that meta-heuristic methods can provide good solutions for medium-sized problems with both renewable and non-renewable resources in a reasonable time. Ballestin *et al.* [7] presented an algorithm for the MRCPSP/Max which chooses the mode first and then uses simulated annealing to solve the resulting RCPSP instance. They also imply that predetermining the modes in the MRCPSP/Max problem can, while reducing complexity, help determine good-quality solutions for the problem. In reference [1], a branch-and-bound algorithm is presented for resource leveling in MRCPSP which yields exact solutions for small-sized instances.

There are other generalizations of the MRCPSP such as energy aware scheduling, speed scaling and resource aware scheduling. Chakrabarti *et al.* [25] considered the problem of finding near-optimal solutions for a variety of NP-hard scheduling problems. They presented improved on-line algorithms for more realistic scheduling models. In reference [34], two techniques for the design of approximation algorithms for NP-hard scheduling problems were presented: The first used a list-scheduling rule while the second yielded on-line algorithms. List scheduling is a method of scheduling by making an ordered list of processes by assigning them some priorities [51]. In reference [55], the authors gave the first constant-factor approximation algorithms for single and parallel machine models. They also presented a list scheduling algorithm for converting preemptive schedules to nonpreemptive ones while only increasing the completion time by a small constant factor. For a set of real-time tasks with precedence constraints executed on a distributed system, Mishra *et al.* [52] proposed static and dynamic power management schemes. In reference [24], α-point scheduling techniques (discussed later) were used to compute approximate solutions for a general class of scheduling problems with each job having a convex non-decreasing cost function. Agrawal and Rao [2] showed that energy-aware scheduling is a generalization of the makespan minimization scheduling problem. Other papers on energy-aware scheduling include references [4, 5, 43, 59].

Kalyanasundaram and Pruhs [42] first introduced the concept of resource augmentation (discussed later). and presented on-line algorithms for two on-line scheduling problems. Phillips *et al.* [56] studied two dynamic scheduling problems and exhibited how to use linear programming relaxations to develop an online algorithm for them. Bansal *et al.* [12] gave an online speed scaling algorithm for the objective of weighted flow time plus energy which also minimizes the wighted flow time subject to an energy constraint. Using resource augmentation, Bansal *et al.* [10] gave approximation algorithms for nonpreemptive minsum scheduling problems (with the objective of total weighted flow time and total weighted tardiness), and presented an integer programming formulation whose relaxation is sufficiently close to the integer optimum.

There are also other papers on speed scaling and scheduling such as Bansal *et al.* [11] where policies for determining the speed of the processor for a minimum energy schedule were studied (the authors showed that no deterministic online algorithm can have a better competitive ratio than the one they presented) and Bansal *et al.* [13] which gave a lower bound on the competitive ratio for the speed scaling problem Bansal *et al.* [14] studied online scheduling problems and considered more general power functions than the one popular in the literature. Nguyen [53] studied online scheduling problems in the resource augmentation/speed scaling models. He presented analyses to prove known facts on the competitiveness of existing algorithms, and also presented competitive algorithms for different settings and objective functions.

In Carrasco [22], the authors allow more general energy cost functions that can be job-dependent in a special setting. They give approximation algorithms for minimizing a scheduling metric and the total energy consumption cost. They use the linear relaxation of the integer programming model to compute a schedule. They introduce the concept of α-speeds, which extend the α-points technique to the case of multiple speeds. They consider "resource cost aware scheduling" (later discussed). For this setting, the authors give more approximation algorithms. They also considerarbitrary speed functions and job-dependent resource cost function. They use resource augmentation techniques in this paper.

"Resource dependent processing times", related to the resource and energy aware scheduling, is also widely studied. Janiak and Kovalyov [39] presented an approximation scheme for a scheduling problem with the processing time of each job linearly dependent on the resource allocated to the job. In [27], a bicriterion problem of scheduling jobs on a single machine was studied. The dependence structure for processing times was similar to that of the previous paper. Grigoriev *et al.* [32] used rounding techniques for parallel machine scheduling with discrete renewable resources. For a more comprehensive review of the literature on resource dependent processing times, the reader is referred to [22] The resource aware scheduling literature also includes [75] (presenting new resource aware scheduling schemes for Hadoop-MapReduce as a popular distributed computing model, [57]) (giving a resource-aware scheduling technique for MapReduce multi-job workloads) and [60] (dealing with time- and volume-dependent cost of resources).

Using a serial SGS, Buddhakulsomsiri and Kim [21] presented a priority rule-based heuristic for the MRCPSP with resource vacations and activity splitting. Their heuristic has problems with the number of activity splitting which increases the solution time. Therefore, it tries to minimize the number of activity splittings In this paper, we mainly continue their line of research, presenting a heuristic which makes use of activity splitting while exhibiting reasonable solution times. We also make use of a simple technique which converts the MRCPSP instance to an RCPSP one, reducing complexity significantly. Here, we assume that all resources are renewable. Our method is compared to two of the existing methods in the literature, *i.e.*, [21,64], and has shown significant advantages over the former while showing acceptable results in comparison to the latter.

In Section 2, we will describe the literature of the MRCPSP in more detail. Note that some of the references mentioned in this paper were conference or working papers which were later published in journals, and there might be slight differences between the published versions and the previous versions used herein.

2. Literature review

Widely studied in the literature, the RCPSP is an NP-hard problem. In general, finding optimal solutions for such problems using a branch-and-bound method involves a huge amount of computation [38,40]. The efficiency of the branch and bound method is hampered as the problem size grows. In particular, due to the complexity of the integer programming model, the method fails to find the optimal solution for problems with tightly constrained resources with 30 activities and five modes within 1000 s [38]. The aforementioned paper shows that using exact methods to solve the integer programming model for the MRCPSP becomes impractical for finding the optimal schedule as the network size grows beyond 50 activities and five modes, especially when resource availability levels are low

There are many papers on the RCPSP, such as Kolisch [44] which considers parallel and serial scheduling methods and provides theoretical results on each method. The author also carries out a computational study

which yields that the parallel method cannot be generally considered as superior. Berthold *et al.* [16] propose a strong hybrid approach for solving the RCPSP with renewable resources, integrating integer programming, constraint programming and satisfiability testing into a branch-and-bound scheme. Their approach improves some previous lower bounds of the PSPlib [46] instances. Bansal and Pruhs [9] present a randomized polynomial-time algorithm for the general scheduling problem with each job having an arbitrary release time and cost function. The objective is to find a preemptive schedule of minimum total cost. Wêglarz [72] deals with continuously divisible and doubly constrained resources. The author assumes the performing speeds to be continuous functions of resource amounts. He also considers the minimum resource consumption ensuring minimum project duration for a given level of resource usage, and the minimum level of resource usage ensuring minimum project duration for a given level of resource consumption. For a survey on the works done on the RCPSP, see [36].

As an extension to the RCPSP, the MRCPSP is strongly NP-hard [41]. Here we present a review on the literature of the MRCPSP and some of its different extensions.

Drexl and Gruenewald [30] present a stochastic scheduling method for the non-preemptive MRCPSPs in which job durations are discrete functions of renewable, nonrenewable and doubly-constrained resources, solving the problems to sub-optimality. Their method was computationally shown to be superior to other deterministic scheduling rules. They also discussed extensions to time-varying job-specific (demand) resource profiles and time-varying supply resource profiles. Kolisch *et al.* [47] consider duration-dependent resource requirements. Sprecher and Drexl [61] propose an exact branch-and-bound method for the MRCPSP which exhibits superior performance and heuristic capabilities.

A simulated annealing approach is presented in [41] for the MRCPSP. Feasible solutions are represented based on a precedence-feasible list of activities and mode assignment. In [20], the authors present a simulated annealing algorithm for the RCPSP and also the MRCPSP. Their efficient method takes into account the specificity of the solution space of scheduling problems.

Zhu *et al.* [78] present a branch and cut algorithm while deriving lower bounds on the distance between each pair of precedence-constrained activities. These bounds are used to reduce the number of variables in the model and to generate cuts that tighten the linear programming relaxation. They also use a branching scheme and a bound-tightening scheme along with a neighborhood search strategy. Their algorithm is exact rather than heuristic in nature. Numerical results were reported for instances of sizes 20 and 30, showing that the solutions found for at least 506 out of the 552 30-activity instances are optimal while others are better than the solutions reported in the literature.

Sabzehparvar and Seyed-Hosseini [58] present a mathematical model for the MRCPSP with mode-dependent time lags and renewable resources. This modeling method has been inspired by the rectangle packing problem. Jarboui *et al.* [40] present a "combinatorial" particle swarm optimization (PSO) algorithm for the MRCPSP, as they note that PSO has not been widely used to solve integer programming problems.

Coelho and Vanhoucke [29] split the problem into two phases of mode assignment and single mode project scheduling, as we do. The former is solved by a satisfiability problem solver while the latter is solved using an existing efficient meta-heuristic procedure for the RCPSP. However, the two phases are executed in one run. Their procedure can yield similar or better solutions than found by other methods, although it often requires a higher CPU time. In reference [1], an exact branch-and-bound algorithm has been presented for resource leveling in the MRCPSP. Instances with up to 12 jobs are solved with an average solution time of about 424 s.

In the literature, heuristic and meta-heuristic methods have been successfully employed to find suitable solutions for the MRCPSP in an acceptable computational time. Heilmann [37] presents a multi-pass priority-rule heuristic method whose performance is evaluated for instances with 100 activities and up to 5 modes. Hartmann [35] presented a genetic algorithm whose encoding was based on a precedence-feasible list of activities and mode assignment. A local search was then used to improve the obtained schedules. Alcaraz *et al.* [3] solve the MRCPSP and its single-mode version with genetic algorithms. In reference [48] parallel and serial SGS's are used and multi-pass priority rule heuristics are designed. In van Peteghem and Vanhoucke [64], a genetic algorithm was presented for both preemptive and non-preemptive MRCPSP, which gave extremely good results for instances of 30, 50 and 100 activities. Their algorithm was known as the most powerful heuristic developed

until 2011 [73] Papers such as [29, 70, 71] have been published after [64] but have not been able to give better results. For example, the results of [71] are not even close to those of [64] in many cases (see [66]). In Section 5.5, we compare our results to some of the results in [64].

In [15], it is shown that genetic algorithms can provide good solutions for problems with 30, 50, and 100 activities, both renewable and non-renewable resources and temporal constraints in 2−4 s. The authors solve the problem in two phases of mode selection and single-mode scheduling, using genetic algorithms for both. This two-phased method is a common way of solving the MRCPSP/Max. For more information, see the literature review in reference [15]. Similarly to the aforementioned paper the same authors in reference [7] solve the MRCPSP/Max in the two phases mentioned. They use simulated annealing for the first phase Their second phase is solving an instance of RCPSP/Max using an existing evolutionary algorithm [6]. Again the authors consider both renewable and non-renewable resources. Their method outperforms the state-of-the-art algorithms for the MRCPSP/Max Ballestin *et al.* [7] also imply that predetermining the modes in the MRCPSP/Max can help both reduce the complexity and determine good-quality solutions for the problem. As it will be seen in Section 5, the same can be mentioned for the MRCPSP. It is worth mentioning that neither [7] nor [15] consider renewable resources in their first phase.

Buddhakulsomsiri and Kim [21] present a priority rule-based heuristic with a serial SGS for the MRCPSP with activity splitting. All resources considered therein are renewable and may be unavailable at some times during the project horizon due to resource vacations. A new concept called moving resource strength (MRS), a dynamic measure of resource tightness, is developed which would lead to more reduction in the project makespan.The concept has been built in the heuristic to control activity splitting during scheduling. The heuristic has problems with the number of activity splitting which increases the solution time significantly. Hence, it tries to minimize this number. Computational experiments demonstrate the effectiveness of the heuristic in reducing the project makespan while minimizing activity splitting. Multiple comparisons have been made on the efficiency of various priority rules. For a more extensive literature review on the MRCPSP, the reader is referred to [73].

In this paper, the main objective is to minimize the project makespan assuming that all resources are renewable and may not be available at all times (due to resource vacations). We propose a new heuristic procedure for solving the MRCPSP by converting it to the single-mode case using a simple technique, and solving the resulting RCPSP instance with a suitable dispatching rule. This conversion reduces the complexity and the solution time for the problem, allowing us to take advantage of activity splitting unlimitedly. The performance of our heuristic is compared with that of reference [21]. We show that better results are obtained in remarkably less computational time using our method. We have been able to solve large-scale problem instances as shown in Section 5 We also compare our results to [64] and show that although their algorithm outperformed ours for tested instances with low complexity, our method yielded good results for higher values of this parameter.

There are also other extensions to the RCPSP and MRCPSP, such as in energy aware and resource aware scheduling, where the former is an example of the latter [22] In these two settings, energy and resource consumptions are also taken into account besides scheduling performance measures [22]. In energy-aware scheduling, for example, for a set of real-time tasks with precedence constraints executed on a distributed system, Mishra *et al.* [52] propose static and dynamic power management schemes in multi-computers. The on-line dynamic power management technique further explores the idle periods of processors. The static scheme can save an average of 10% more energy. When combined with dynamic schemes, energy savings is significantly improved.

Goiri *et al.* [31] present a dynamic job scheduling policy for energy-aware resource allocation in a datacenter. This policy tries to integrate workloads from separate machines into a smaller number of nodes while satisfying the amount of hardware resources needed to fulfill the service level. This allows turning off the extra servers which reduces the overall power consumption. Young *et al.* [76] study the problem of energy-constrained static resource allocation of a collection of communicating tasks to a computing environment. They maximize the probability that the collection of tasks completes by a deadline and that an energy constraint is satisfied. They consider task execution times and communication times to be stochastic. They design and evaluate a set of heuristics for allocating resources in their system. Their approach can improve the performance of the aforementioned system.

Scheduling of tasks on a multi-machine system to reduce the makespan, while satisfying the precedence constraints between the tasks, is known to be an NP-hard problem, similarly to the RCPSP. Agrawal and Rao [2] propose a new formulation and show that energy-aware scheduling is a generalization of the makespan minimization scheduling problem. They propose three effective algorithms for energy-aware scheduling. The first is a genetic algorithm that searches for an energy reducing schedule. The second uses cellular automata to generate low energy schedules, while using a genetic algorithm to find good rules for the cellular automata. The third is a heuristic which gives preference to high-efficiency machines in allocation.

Some other approaches have been used in energy- and resource- aware scheduling such as integer relaxations of models, α-points, and list scheduling algorithms. In reference [34], two techniques for the design of approximation algorithms for NP-hard scheduling problems with the objective of minimizing the weighted sum of job completion rimes are given. The first approach uses an optimal solution to a linear programming relaxation in order to guide a list scheduling rule while the second yields on-line algorithms. The approach of applying a list scheduling rule in which jobs are ordered based on solving a linear program can easily be extended to a wide range of scheduling problems. Preemption has also been considered, and the concept of α-points is used: the α-point of a job is defined as the earliest time at which an α fraction of the job has been completed in the linear relaxation of the integer scheduling problem [24]. This notion is used to derive an upper bound for the expected completion time of the jobs. In reference [56], the objective is to minimize the average completion time of the set of arriving jobs. The authors give the first constant-factor approximation algorithms for preemptively scheduling parallel machines with release dates. Many of the algorithms presented in [55] are based on relationships between preemptive and nonpreemptive schedules and linear programming relaxations of both. The authors use a list scheduling algorithm which converts preemptive schedules to nonpreemptive ones while only increasing the total (weighted) completion time by a small constant factor. These results give some insight into the power of preemption (the improvement preemption makes in the average completion time).

Phillips *et al.* [54] consider the problem of scheduling some jobs with release dates on identical parallel machines; they present a list scheduling algorithm, and prove that the ratio of the average completion time of the optimal nonpreemptive schedule to that of the optimal preemptive one is at most $\frac{7}{3}$. They also give an improved bound on the quality of the linear programming relaxation of the problem given in [34]. Chakrabarti *et al.* [25] consider the problem of finding near-optimal solutions for a variety of NP-hard scheduling problems. They give improved bounds on the power of preemption in scheduling jobs with release dates on parallel machines. They present improved on-line algorithms for more realistic scheduling models, such as environments with parallelizable jobs, jobs contending for shared resources, *etc.* The authors improve and extend the same technique of list scheduling in the order determined by solving a linear program. They also improve a result of reference [33], an earlier version of reference [34].

Carrasco *et al.* [23] present several new approximation algorithms for energy aware scheduling problems with general job-dependent energy cost functions in the non-preemptive setting. They also consider maintenance and replacement costs. They extend their algorithms to approximating weighted tardiness, a problem that had not been addressed in the literature before. Their methodology extends the idea of α-points to the energy aware setting by developing the α-speeds concept, related to speed scaling (to be discussed in more detail in a few paragraphs).

In reference [24], resource augmentation (to be discussed in more detail later) and α-point scheduling techniques have been combined to compute approximate solutions for a general class of scheduling problems, yielding a very good performance. Each job has a convex non-decreasing cost function and the goal is to compute a schedule that minimizes the total cost subject to arbitrary precedence constraints Without loss of generality, scheduling is done non-preemptively.

The concept of α-points has also been used in [8], where MapReduce, a prominent programming model, is used for power aware scheduling under a given budget of energy. The authors present a polynomial time approximation algorithm. They propose a convex programming formulation to combine with list scheduling policies. Other papers on energy-aware scheduling include [5] (scheduling of periodic tasks to reduce CPU energy consumption through dynamic voltage scaling and computing the optimal speed [43]), (energy-efficient

scheduling of periodic real-time tasks with deadlines on chip multi-core processors with an algorithm that determines a globally energy-efficient frequency and saves up to 55% more power compared to an existing method) [59], (presenting a methodology to obtain an energyoptimal voltage assignment) and [4] (proposing an energy aware, offline, probability-based scheduling algorithm for multiprocessor systems, to minimize the number of processors used, maximize the utilization of the processors, and optimize the energy consumption).

Now we study the literature of speed scaling in detail Nowadays, microprocessors chips are produced with dynamically scalable speeds in order to save energy. In the last few years there has been a great deal of research on scheduling problems that arise in this issue. Generally we want to optimize both a scheduling quality of service and a powerrelated criterion. Scheduling algorithms for such problems have two parts: a job selection policy to determine which job to run and a speed scaling policy to determine the speed at which the processor is run. Almost all of the theoretical speed scaling research has assumed that the power function, which expresses the power consumption P as a function of the processor speed s, is of the form $P = s^{\alpha}$, where $\alpha > 1$ is some constant. Bansal *et al.* [14], however, consider more general power functions. They investigate online scheduling problems with speed scaling where the speeds may be discrete. They develop competitive algorithms for the problem minimizing the total or fractional flow (the time lag between when a job is released to the system and when it is done) plus energy. They also allow preemption in their setting.

Bansal *et al.* [12] state that an operating system must have a speed scaling policy. They give an online speed scaling algorithm for the objective of weighted flow time plus energy. This algorithm also efficiently constructs a schedule for minimizing the weighted flow time subject to an energy constraint. There are also other papers on speed scaling and scheduling such as [13] which gives a lower bound on the competitive ratio for this problem, and [11] which studies policies for determining the speed of the processor for a minimum energy schedule. The authors provide a tight bound on the competitive ratio of a previously presented algorithm with respect to energy. They then propose a new online algorithm and show that no deterministic online algorithm can have a better competitive ratio.

A concept used in speed scaling is resource augmentation which we describe here in detail The idea of resource augmentation is that we compare the optimal solution to the output of the algorithm considering we have additional resources [22]. Kalyanasundaram and Pruhs [42] introduce this concept and apply it to two on-line scheduling problems: the uniprocessor CPU scheduling and the best-effort firm real-time scheduling problems. They show that there are simple on-line scheduling algorithms for these problems. Phillips *et al.* [56] study two problems in dynamic scheduling - scheduling to meet deadlines in a preemptive multiprocessor setting and scheduling to provide good response time in some scheduling environments. The authors show how to use linear programming relaxations to develop an online algorithm for scheduling a single machine preemptively. Some theoretical results for list scheduling algorithms are also presented. Moreover, the authors study existing algorithms and provide optimality conditions for them. This paper gives the first results in the literature for optimization of the average flow time. Again, using resource augmentation, Bansal *et al.* [10] give polynomial-time approximation algorithms for several nonpreemptive minsum scheduling problems where jobs arrive over time and must be processed on one machine. The authors present an integer programming formulation the relaxation of which is sufficiently close to the integer optimum. Nguyen [53] studies online scheduling problems in the resource augmentation/speed scaling models. He considers a possibly non-convex relaxation and uses its Lagrangian dual to have an objective value within a factor of the primal optimum. He proves some known facts on the competitiveness of some existing scheduling algorithms. He also considers different settings and objective functions and presents some competitive algorithms.

In Carrasco [22], the problem of scheduling with non-renewable resources is considered; in special settings, the authors allow more general energy cost functions that can be job-dependent. They give approximation algorithms for minimizing a scheduling metric and the total energy consumption cost. They use the solution of the linear relaxation of the integer programming model to compute a schedule. They introduce the concept of α-speeds, which extend the α-points technique to the case of multiple speeds. Their model considers job-dependent energy costs and they further generalize the results to the setting where multiple resources are available, and the consumption level of all resources determine the speed at which jobs are processed. This is called “resource

cost aware scheduling" For this setting, the authors give approximation algorithms for minimizing a scheduling metric and resource consumption cost. They also generalize the resource dependent processing times literature by consideringan arbitrary speed function. They also consider an arbitrary job-dependent resource cost function The authors also use resource augmentation in this paper for some of the algorithms.

"Resource dependent processing times" is a topic directly related to the resource and energy aware scheduling problems, but it was developed independently [22]. Janiak and Kovalyov [39] study the problem of scheduling jobs on a single machine. The processing time of each job is a linear decreasing function of the amount of a common resource allocated to the job. An approximation scheme is presented for the problem with discrete resource. In [27], a bicriterion problem of scheduling jobs on a single machine is presented, with the maximal or total (weighted) resource consumption and a regular scheduling metric as the two criteria. The processing times have similar dependence structure as the previous paper. General schemes for the construction of the Pareto set are presented. There are also papers such as [32] considering discrete renewable resources for parallel machine scheduling and using linear programming rounding techniques − based on relaxation of the problem − to allocate resources to jobs and assign jobs to machines. The authors also use list scheduling to present an approximation algorithm for the problem Cheng *et al.* [28] present a review on resource dependent processing times.

There are many other papers in resource aware scheduling, such as [68] (proposing distributed scheduling algorithms for large volumes of arbitrarily divisible loads being processed at cluster/grid systems [75]), (presenting new resource aware scheduling schemes for Hadoop-MapReduce as a popular distributed computing model [57]), (giving a resource-aware scheduling technique for MapReduce multi-job workloads − the technique is guided by user-provided completion time goals for each job) and [60] (dealing with resource cost aware scheduling and time and volumedependent cost of resources; lower bounds[2] on the cost and different linear and mixed integer programming models as well as greedy algorithms are presented). More examples of resource-aware scheduling can be found in reference [63, 77].

The rest of this paper is organized as follows: in Section 3, we present the mathematical model of the MRCPSP with activity splitting and describe the erroneous constraints therein. Section 4 describes the proposed heuristic solution procedure. Section 5 shows the computational experiments which comprise solving small instances to optimality, comparing our method to that of reference [21], solving very large-sized instances, implementing our method for standard instances, comparing it to the method in [64] and validating the two-phased structure of our method. Finally, Section 6 concludes the paper.

3. Problem description and corrected model

In this section, we first present a mixed integer programming formulation for the MRCPS with activity splitting and then correct some erroneous mistakes in it.

3.1. Notations and Problem formulation

Let us assume that a project consists of J activities (jobs), labeled with $j = 1, 2, 3, \ldots, J$. The activities are shown on nodes (AON representation), and arcs denote the precedence relations. There is a unique dummy source (start activity) with $j = 1$ and a unique dummy sink (end activity) with $j = J$. In essence, each activity cannot be started unless all its predecessors have been finished and all the required resources are available for the duration of the activity. This assumption is too restrictive and may lead to extension of the project makespan. If activities can be split when resources are unavailable, the resulting schedule could be significantly improved. It is worth mentioning that splitting and preemption should be treated differently. Pausing an ongoing activity because of resource unavailability and resuming it later may have small financial or time impact. However, interrupting an ongoing activity by switching to another activity can result in a high penalty such as setup time lost, re-configuring complicated equipment settings, *etc.* A preempted activity is an activity for which there

[2]Used for bounding or pruning during optimization.

is at least one time period after the start of the activity wherein the activity is eligible (resource feasible for renewable resources and precedence feasible) to be processed but is not being processed. However, a split activity is an activity that is not processed in consecutive time periods. A preempted activity is a split activity, but the converse is not necessarily true. The cases wherein activities are split may result from insufficient resources rather than by choice [26]. In general, activity splitting could considerably reduce the project makespan when resources are scarce. Buddhakulsomsiri and Kim [21] note that activity splitting poses no additional duration. Mode switching is not allowed when activities are resumed after splitting.

In the proposed algorithm, we take advantage of activity splitting in discrete time whenever there are not enough resources to complete an activity. The availability schedule of each renewable resource is known in advance. Each activity j can be processed in one of M_j possible modes. The total duration of each activity in each mode is fixed and is not increased due to splitting.

The notations used in this paper are as follow:

J : The total number of activities of the project.
M_j : The number of modes of activity $j, j = 1, \ldots, J$.
R : The number of renewable resources of the project.
d_{jm} : The duration of activity j when executed in mode m, $j = 1, \ldots, J, m = 1, \ldots, M_j$ which is an integer.
T : The upper bound for the project makespan, *i.e.*, $T = \sum_{j=2}^{J-1} \max_{m=1}^{M_j} d_{jm}$.
P_j : The set of all activities that precede activity $j, j = 1, \ldots, J$.
S_j : The set of all activities that succeed activity $j, j = 1, \ldots, J$.
ES_j : The earliest start time of activity $j, j = 1, \ldots, J$.
LF_j : The latest finish time of activity j, obtained through the critical path method (CPM) calculations by setting $LF_J = Tj, j = 1, \ldots, J$.
k_{jmr} : The amount of resource of type r required per period to execute activity j in mode m, $j = 1, \ldots, J, m = 1, \ldots, M_j, r = 1, \ldots, R$.
K_{rt} : The amount of resource of type r available for period t, $r = 1, \ldots, R, t = 1, \ldots, T$.
M : A very large positive number.

The decision variables are:

$$x_{jmt} : \begin{cases} 1, \text{if job } j \text{ is executed in mode } m \text{ in period } t, \\ 0, \text{ otherwise.} \end{cases}$$

$$y_{jm} : \begin{cases} 1, \text{if job } j \text{ is executed in mode } m, \\ 0, \text{ otherwise.} \end{cases}$$

ST_j : The start time of activity j
FT_j : The finish time of activity j

Mixed Integer mathematical programming models have been developed in many references such as [21] or [47]. Using the aforementioned notations, the mathematical model as expressed in [21] − is as follows:

$$\min FT_J \tag{3.1}$$

$$s.t.$$

$$\sum_{m=1}^{M_j} y_{jm} = 1 \quad \forall j \in \{1, \ldots, J\} \tag{3.2}$$

$$\sum_{t=ES_j}^{LF_j} x_{jmt} = d_{jm} \times y_{jm} \quad \forall j \in \{1, \ldots, J\}, \forall m \in \{1, \ldots, M_j\} \tag{3.3}$$

$$x_{jmt} \times t \leqslant FT_j \quad \forall j \in \{1, \ldots, J\}, \forall m \in \{1, \ldots, M_j\}, \forall t \in \{ES_j, \ldots, LF_j\} \tag{3.4}$$

$$x_{jmt} \times t + (1 - x_{jmt}) \times M \geqslant ST_j \quad \forall j \in \{1, \ldots, J\}, \forall m \in \{1, \ldots, M_j\}, \forall t \in \{ES_j, \ldots, LF_j\} \tag{3.5}$$

$$FT_i \leqslant ST_j - 1 \quad \forall j \in \{1, \ldots, J\}, \forall i \in P_j \tag{3.6}$$

$$\sum_{j=2}^{J-1} \sum_{m=1}^{M_j} k_{jmr} \times x_{jmt} \leqslant K_{rt} \quad \forall r \in \{1, \ldots, R\}, \forall t \in \{1, \ldots, T\} \tag{3.7}$$

$$x_{jmt} \in \{0, 1\} \quad \forall j \in \{1, \ldots, J\}, \forall m \in \{1, \ldots, M_j\}, \forall t \in \{ES_j, \ldots, LF_j\} \tag{3.8}$$

$$y_{jm} \in \{0, 1\} \quad \forall j \in \{1, \ldots, J\}, \forall m \in \{1, \ldots, M_j\} \tag{3.9}$$

$$ST_j, FT_j \geqslant 0 \quad \forall j \in \{1, \ldots, J\}. \tag{3.10}$$

The objective function − equation (3.1) − minimizes the makespan of the project. Constraints (3.2) show that activity j is executed in one and only one mode. Constraints (3.3) assure that activity j is executed only one time, and the total number of periods in which activity j is being executed is equal to its duration in mode m. Constraints (3.4) and (3.5) show the finish and start times of activity j constraints (3.6) represent the precedence relations and constraints (3.7) are capacity constraints.

In Section 5.2, we will solve a number of instances of the model (3.1)−(3.10) in order to evaluate the solution times needed for solving the integer programming model of the MRCPSP.

3.2. The corrected model

The model contains some erroneous constraints since it disregards a point about the last dummy activity J. Since the duration of this activity is zero in all modes, constraints (3.3) will yield zero for each $x_{Jmt}, m = 1, \ldots, M_J, t = 1, \ldots, T$. Therefore, based on constraints (3.4), FT_J will also be zero as a result of the fact that the objective function is the minimization of FT_J. Hence, we get a makespan of zero. Similarly, because of constraints (3.5) and again all the x variables related to J being equal to 0, we have $ST_J \leqslant M$. While FT_J is equal to zero, ST_J goes to its upper bound M. It even drags the finish time of direct predecessors of J to $M - 1$ (due to constraints (3.6)). In order to avoid having a makespan of zero, we change the objective function (3.1) to ST_J, and exclude the dummy activities from constraints (3.3)−(3.5) so that the model yields correct answers. There are other ways of correction too, but this was the easiest suggested by one of the reviewers. Moreover, in order to avoid computational problems with the index t in constraints (3.4) and (3.5) − while there may not be any − we start the scheduling from $t = 1$.

We implemented the model in GAMS for instances of size 16 in the PSPlib [46]. Without the correction, the yielded makespan is equal to zero and for some instances, the start and finish times of activities are different from what they would be with the correction; *i.e.*, for some instances, the yielded start and finish times will be incorrect without the correction. One should notice that the corrections do not affect the heuristic procedure presented in reference [21].

4. The proposed solution procedure

In this paper, we present a new heuristic solution procedure (NHSP) which can solve very large problem instances consisting of several thousand activities. In order to assess the performance of the heuristic method, an analysis of its computational time as well as the quality of the solutions found is very important. Our central idea for obtaining a good solution for project scheduling is to allow activity splitting while keeping the solution time reasonable. This could lead to a major reduction in the project makespan and better usage of resources. Another key idea used herein is to convert the problem to the single mode case, *i.e.*, an instance of RCPSP which would highly reduce the solution time. To do this, we must determine one mode out of M_j possible modes for each activity j. A simple technique is presented in the next subsection for this issue. This idea of conversion is validated and justified in Section 5.5.

4.1. Converting to a single mode problem

In the proposed solution procedure, the conversion from MRCPSP to RCPSP is done first. To select an activity mode, a weight will be calculated for each mode. These weights specify the proportion of utilized resources for each activity and each mode. The main point of having these weights is to project the limitations of resources. The single mode will be selected such that the usage of resources with less availability is minimized. In this paper, these weights are denoted by ω_{jm} (weight of activity j in mode m, $\forall j = 1, \ldots, J, m = 1, \ldots, M_j$) and are computed as follows:

$$AR(r) = \frac{\sum_{t=1}^{T} K_{rt}}{T} \tag{4.1}$$

$$RR(r) = \frac{\sum_{j=1}^{J} \sum_{m=1}^{M_j} k_{jmr}}{\sum_{j=1}^{J} M_j} \tag{4.2}$$

$$WR(r) = \frac{RR(r)}{AR(r)} \xrightarrow{Normalized} NWR(r) = \frac{WR(r)}{\sum_{r=1}^{R} WR(r)} \tag{4.3}$$

$$\omega_{jm} = \sum_{r=1}^{R} \left(d_{jm}^2 \times k_{jmr}^{NWR(r)} \right). \tag{4.4}$$

Equation (4.1) shows the average amount of resource r that is available per period. Equation (4.2) gives the average of resource requirements of activities in different modes equation (4.3) estimates the weight of resource r which is the proportion of the resource requirements to resource availability ((4.2) divided by (4.1)). WR is normalized and changed to NWR; therefore it will be between 0 and 1. More availability of each resource or a less required amount of it yields a smaller value of $NWR(r)$. Equation (4.4) calculates the weights for each mode of jobs. The single mode for activity j is selected as the minimum value of $\omega_{jm}, \forall j = 1, \ldots, J, \forall m = 1, \ldots, M_j$. After performing mode selection for each activity, the problem is converted to an instance of RCPSP.

We will see in Section 5 that this simple conversion technique yields good results. However, in 3 out of 640 instances of the PSPlib [46] for size 30, the resulting RCPSP is infeasible in the sense that the amount of resources needed in that mode cannot be available in any time for at least one activity. Therefore, a correction to this method is needed. We implement this correction by using a notion similar to the Shortest Feasible Mode (SFM) rule [18] which is shown to be one of the most robust mode selection rules [21][3]. In the aforementioned

[3]As it appears, the authors in [21] have used the SFM rule during scheduling, not as a mode selection rule in order to convert the problem to the single mode case, as we do. For more information, see [21].

three instances, the selected mode is shifted to the next shortest feasible mode until the resulting RCPSP is feasible in the sense mentioned.

For a full analysis of this mode selection rule, see Section 5.5.

4.2. Dispatching rules

Buddhakulsomsiri and Kim [21] developed three priority rule-based heuristics that include some activity priority and mode priority rules for solving the MRCPSP. The three heuristics applied therein are:

- deterministic multi-pass priority rule-based heuristics (DM),
- stochastic multi-pass priority rule-based heuristics with biased random sampling (BRS) and,
- stochastic multi-pass priority rule-based heuristics with regret-based random sampling (RBRS).

For more information on these heuristics, the reader is referred to [21]. The heuristic proposed herein uses a dispatching rule for scheduling the activities in the RCPSP while allowing preemption. This heuristic must specify an activity to be performed in each period of time. A decision point (t) is defined for each activity in which that activity will be checked for the possibility of scheduling. A set of eligible jobs is determined at any decision point composing of those activities which can be started at that time, meaning that all their predecessors have been done and their required resources are available at least for that time.

To start scheduling, the decision point is initially set to $t = 1$. In a decision point (t), if there is no activity that can be scheduled, we move to the next decision point $(t+1)$. To choose an activity for scheduling from the set of eligible jobs, dispatching rules are employed. This work examines the performance of three dispatching rules for scheduling an RCPSP instance with splitting. One of them is WLST (weighted latest start time) that was used in [69] and the two others are GNS (Greatest number of successors) and RPW (Rank positional weight) which were examined in [74] These dispatching rules are described as follows:

- WLST : Priority is given to the activity that has the largest value of weighted latest start time

$$WLST = \frac{1}{(LS_j + d_j)}. \tag{4.5}$$

- GNS: Priority is given to the activity that has the largest number of successor activities.
- RPW: Priority is given to the activity whose duration plus the durations of all its successors has the largest value.

Using the dispatching rules above one job from the eligible jobs set (EJ) is selected and is called a selected job (SJ). Then, this selected activity is evaluated in terms of how far it can be executed based on resource availability. The activity is split if there are not enough resources. The duration of the split activity is then changed to the remaining duration.

We do not impose any limitation on the number of activity splitting unless physical constraints force us to do so. Activity splitting leads to a better utilization of available resources (this becomes increasingly important when resources are expensive) and could entail a significant reduction of the project makespan. For those activities with high setup costs, the number of allowable splitting could be limited in the algorithm.

4.3. Statement of the proposed algorithm

We summarize our proposed solution procedure in Figure 1 (besides the notations of the model (3.1)–(3.10), consider the following: m_j is the mode chosen for activity $j = 1, \ldots, J$ through (4.4), DP_j is the decision point of activity $j = 1, \ldots, J$, EJ is the set of eligible jobs determined at each point $t = 1, \ldots, T$ (an index t is necessary but is omitted for the sake of brevity), $\tilde{K}_r(t) = K_{rt} - \sum_{j \text{ is being executed in t}} k_{jmr}$ is the remaining amount of resource r at time $t, \forall r \in 1, \ldots, R, \forall t \in 1, \ldots, T$, $NEJ(t)$ is the set of activities that are not eligible in $t = 1, \ldots, T$ (some activities may be added to this set in each iteration), $SchedAct$ is the set of activities that are already scheduled and $Succ_j$ is the set of the successors of activity $j = 1, \ldots, J$ – the notations and

presentation of the algorithm are inspired by the one in [21]):

Initialization: calculate w_{jm}, $m_j = \min_m\{w_{jm}\}$, $\forall j = 1, \ldots, J$,

$t = 1$, $SJ = 1$, $\quad DP_j = \max\{t + 0, DP_j\}, \forall j \in Succ_1$,

while $j < J$,

If $j \notin SchedAct$,

$EJ = \{EJ \cup \{j\} | DP_j \leqslant t\}, EJ = EJ \backslash NEJ(t)$

If $EJ = \varphi$,

$t = t + 1$, $\quad$ Continue

End

Select $SJ \in EJ$ using a dispatching rule

For $t' = t$ to $t + d_{SJ}$,

If $k_{(SJ)mr} \leqslant \tilde{K}_r(t')$, $\forall r \in 1, \ldots, R$,

Schedule SJ without splitting, $DP_j = \max\{t + d_{SJ}, DP_j\}, \forall j \in Succ_{SJ}$

$\tilde{K}_r(t') = \tilde{K}_r(t') - k_{(SJ)r}, \forall r \in 1, \ldots, R$

Else if $\exists r \in 1, \ldots, R : k_{(SJ)mr} > \tilde{K}_r(t')$ and SJ cannot be split,

$EJ = EJ \backslash \{SJ\}, NEJ(t) = NEJ(t) \cup \{SJ\}$, $\quad$ break

Else if SJ can be split,

Schedule SJ up to when enough resources exist; change the duration to the remaining duration, $\tilde{K}_r(t'') = \tilde{K}_r(t'') - k_{(SJ)r}, \forall r \in 1, \ldots, R, \forall t''$ until enough resources exist

End

End

If $SJ \in NEJ(t)$,

If $EJ = \varphi$

$t = t + 1$, $\quad$ Continue

End

End

End

End

Figure 1. The proposed solution procedure.

5. Computational results

In this section, we present the computational results of our investigations. In Section 5.1, the parameter settings of the generated test problems are explained. In Section 5.2, we try to solve some of these generated test problems to optimality using GAMS. Sections 5.3 and 5.4 exhibit our computational results and compare them to those of [21] for medium and large-sized instances. For large-sized instances, no comparison is possible since Buddhakulsomsiri and Kim [21] did not report any results on such instances. Section 5.5 implements our method for instances of standard libraries such as the PSPlib [46], the Boctor multi-mode dataset [17] and the MMLIB library [66]. Finally, Section 5.6 verifies the two-phased nature of our method.

5.1. Generated problems

In order to compare our results to those of reference [21], some random test problems were generated using a modified version of ProGen developed by Kolisch *et al.* [47]. Resource vacation parameters mentioned in [21] have also been added to the test problems and thus, our results are comparable to those of that paper. For this comparison, the instances found in standard libraries such as the PSPlib [46] are not eligible since they do not

TABLE 1. PROGEN assumptions.

	NC	RF	RS	VP	VL
Min	1.5	0.5	0.25	0.1	0
Max	2.0	1	0.5	0.3	1

TABLE 2. Other problem assumptions 1.

	MODES	Duration	Short vacation range	Long vacation range
Min	4	1	1	3
Max	4	10	2	5

TABLE 3. Other problem assumptions 2.

	# of resources	resource requirement	# of activity1 successors		# of activity j successors		# of activity J predecessors	activity j predecessors
Min	4	1	2	1	2	1		
Max	4	4	5	5	5	5		

match the instances of Buddhakolsomsiri and Kim [21] in terms of parameter settings. Testing our method on these standard instances is left for Section 5.5.

A brief description of each modified ProGen parameter is given below:

- Network complexity (NC) is a measure of the complexity of the precedence relations, calculated as the average number of non-redundant arcs per node.
- Resource factor (RF) is the average number of resources which are required per activity. If it is normalized, $RF = 0$ shows that no activities require any resources while $RF = 1$ implies that each activity requires the same amount of each resource.
- Resource strength (RS) is a measure which indicates the tightness of resources. Again, when we normalize it to be between 0 and 1, $RS = 0$shows that resource availability is the minimum amount guaranteeing the existence of a feasible schedule while $RS = 1$ indicates that resource availability is not a constraint. We will see in Section 5.5 that if the resource availability is excessive, the value of this parameter can be more than 1.
- The percentage of resource unavailability (VP) is the percentage of total possible resources that are unavailable (because of vacation).
- The vacation length factor (VL) exhibits the density of vacations. VL is also between 0 and 1. A small VL value indicates that there are more frequent but shorter resource vacations.

These parameters, along with the parameters related to the size of instances and other parameters, are set to the same amounts as those of Budhakolsomsiri and Kim [21] (shown in Tabs. 1−3. The first table is about ProGen parameter settings and the other two tables are for the other parameters) It is also worth mentioning that the network complexity can also be described by the order strength, which is defined as the number of precedence relations divided by the theoretical maximum number of precedence relations [49, 65].

We generated some random problem instances with parameters between the minimum and maximum levels given in the three tables above. The numbers of activities are set to 30, 60 and 90. The numbers of instances with 30 activities are 500. Because of the computational time needed for problem generation the numbers of instances with 60 and 90 activities are decreased to 200, as the authors in [21] did These instances are then solved using our heuristic method in order to be compared to the method in [21].

TABLE 4. Summary of computation time (s).

# of instanc	30		60			
	Multimod		Single mode (converted through (4.4))		Single mode (converted through (4.4))	
	Computation time	Makespan	Computation time	Makespan	Computation time	Makespan
1	–	–	6.325	39	–	46 (85.89%)
2	–	–	7.273	44	–	62 (83.11%)
3	–	–	12.678	43	–	–
4	–	–	8.821	18	–	40 (87.94%)
5	–		114.64	41	–	28 (91.42%)
6	–	–	8.157	28	out of memor	–
7	–	–	59.159	22	–	–
8	–	–	21.711	41	–	–
9	–	–	8.565	28	–	47 (84.57%)
10	–	–	52.786	22	–	–
11	out of memor		589.691	25		
12	–	–	–	–		
13	–	–	16.567	35		
14	–	–	15.552	38		
15	–	–	29.711	36		
16	–	–	7.429	28		
17	–	–	47.663	22		
18	–	–	43.239	51		
19	out of memor	–	537.613	25		
20	–	–	18.765	41		
Averag	–	–	84.54447368	33	–	–

5.2. Exact solutions

It was already mentioned that the (M) RCPSP is (strongly) NP-hard. Before testing our heuristic method, we wish to evaluate the computational time needed for solving the MRCPSP to optimality. Therefore, we implement the optimization problem (3.1)−(3.10) in GAMS 23.6 with CPLEX 12 as solver Sets of instances generated using the modified ProGen were solved for this purpose. The results of the solution times and objective functions for 20 instances of size 30 and 10 instances of size 60 are given in Table 4 (note that a hyphen means that the corresponding instance did not converge to optimality within 1200 s). Since no optimal solutions for instances of sizes 30 and 60 were found, there is no need to test instances of size 90.

As it can be seen, the optimal solution to none of the MRCPSP instances can be found in a reasonable time. Based on the advice of one of the reviewers, in order to evaluate the results obtained by our heuristic − regardless of the mode selection method − the aforementioned generated instances are solved with GAMS for the optimal modes selected through (4.4). The average solution time for instances of size 30 is about 84 s, and 19 out of 20 instances reach optimality. This average solution time is large compared to the average solution times of our heuristic method (see Sects. 5.3 and 5.4) for instances of size 30. For instances of size 60, the converted RCPSP is still impossible to solve within 1200 s; it was only able to find feasible solutions with large optimality gaps (shown in percentages in Tab. 4) in 5 out of 10 instances. Therefore, our algorithm outperforms this method (solving the single-mode instance converted through (4.4)) in terms of computational time for instances of size 60 and more, at least when resource vacations are present We also aim to evaluate the quality of the solutions derived through our heuristic method, *i.e.*, the average deviation of the solution from the lower bound of the

TABLE 5. Summary of the project makespan.

# of activity	Proposed Heuristic			Competing Heuristic		
	WLST	GNS	RPW	DM	BRS	RBRS
30 (Average)	49.0	50.3	49.8	55.8	46	46.2
	45.5	46.6	46.1	50.5	43.2	42.9
No split (Max)	95	94	91	–	– –	–
Split (Max)	80	84	94	–	– –	–
60 (Average)	63.5250	63.8900	64.8200	101.3	82.2	83.9
	56.3950	57.4700	59.3700	90.7	77.7	78.3
No split (Max)	110	110	118	–	– –	–
Split (Max)	98	95	110	–	– –	–
90 (Average)	69.5500	69.0400	72.0700	128.2	104.5	105.5
	60.5800	61.1900	65.8250	117.9	102.7	102.7
No split (Max)	119	110	122	–	–	–
Split (Max)	101	89	120	–	–	–

problem (which is found by considering minimal durations in the CPM). This is thoroughly investigated in Section 5.5.

We would also like to analyze what the drawback of predetermining the modes is. This issue is detailed in Section 5.6.

5.3. Medium-sized problems

In this subsection, computational experiments are collected and compared in two categories. The first set is the proposed heuristic of this paper, which is evaluated in terms of the project makespan, computational time and number of activity splitting using three dispatching rules (WLST, GNS and RPW). Another set is the competing heuristics, which are the results of [21].

In Table 5, summary of the results for the project makespan is shown which is the main objective of the proposed solution procedure. The results of the proposed heuristic are comparable to the best results of the competing heuristics (RBRS) and are even better when the number of activities has been increased For each size 30, 60 and 90, the results are given in two rows: with and without splitting the activities (the first row is without splitting). The worst case results (the maximum objective function values) are also given.

It is worth noting that since in both the optimization model (3.1)−(3.10) and in [21], the latest start time of activity J is set equal to the upper bound T, we tried to do the same in our heuristic procedure. However, setting the latest start time of J equal to its earliest start time decreased the makespan. Therefore, we preferred the second way.

The results shown in Table 5 indicate that the best makespan values in instances with 30 activities are obtained by using the RBRS. Comparing our results with that heuristic shows that the results for project instances with 30 activities are very close to those of the RBRS and get much better as the number of activities increases to 60 and 90.

We can understand from the results in Table 4 that preempting generally reduces the makespan when no nonrenewable resources are considered, as reported by reference [64].

Computational time is also a significant criterion for evaluating the proposed heuristic. The results for this performance measure are reported in Table 6.

The proposed heuristic both with and without splitting was run on MATLAB on a personal computer with a processor core 2 Duo at 2.8 GHz. The average computation times shown in Table 6, indicate that the solution times of the proposed heuristic are negligible when compared with those of the competing heuristics, and of

TABLE 6. Summary of computational time (s).

# of activity	Proposed Heuristic			Competeing Heuristic		
	WLST	GNS	RPW	DM	BRS	RBRS
30 (Average)	0.011	0.009	0.012	1.33	62.93	63.08
	0.012	0.010	0.013	3.12	156.09	153.56
No split (Max)	0.0275	0.022	0.0266	1.52	70.56	71.00
Split (Max)	0.028	0.0261	0.0311	3.94	203.36	195.92
60 (Average)	0.0293	0.0267	0.0300	4.02	189.69	189.29
	0.0351	0.0335	0.0368	11.08	567.63	568.38
No split (Max)	0.0810	0.0549	0.0520	4.95	224.83	224.51
Split (Max)	0.0967	0.0616	0.0770	17.52	813.25	842.39
90 (Average)	0.0415	0.0393	0.0455	8.09	378.75	382.03
	0.0490	0.0471	0.0555	24.99	1252.1	1263.17
No split (Max)	0.0556	0.0590	0.0838	9.30	423.83	430.39
Split (Max)	0.0809	0.0794	0.0997	40.56	2003.09	2074.11

TABLE 7. Summary of the average number of activities splitting.

# of activity	Proposed Heuristic			Competeing Heuristic		
	WLST	GNS	RPW	DM	BRS	RBRS
30 (Average)	3.09	3.02	2.94	4.6	4.5	4.5
Max	10	14	8	–	–	–
60 (Average)	7.1250	6.8450	6.6550	12.3	13	13.9
Max	16	15	18	–	–	–
90 (Average)	11.1450	10.3850	10.6950	19.3	21.2	20.4
Max	23	25	23	–	–	–

course with those of GAMS The computation times for instances of sizes 60 and 90 are less than 0.1 s while the minimum average time taken by the competing heuristics is 4.02 s and the value found for the makespan is still poor. It is also worth mentioning that the maximum of the computational times for our heuristic (0.0997 s) is less than the minimum of the computational times of the competing heuristics (1.33 s) for all sets of 30, 60 and 90.

One last comparison we would like to highlight is on the average number of activity splitting which is given in Table 7.

The average number of activity splitting done in the competing heuristics is more than that in the proposed heuristic. In our heuristic, each activity can be split whenever it is necessary for scheduling and we do not need to force any limitations on the number of activity splitting since the proposed algorithm is fast.

To sum up, each of these heuristics has some advantages and some shortcomings. However, the proposed heuristic outperforms the competing heuristics by taking advantage of splitting, in terms of the project makespan (mainly in instances with 60 and 90 activities), computational time and number of activity splittings.

5.4. Solving large scale problems

As seen in Section 5.3, our method tends to perform better as the number of activities grows larger. This motivates us to implement our method for even larger instances, *i.e.*, those with 100 to 5000 activities. Table 8 gives the results for randomly generated instances solved with the proposed heuristic using the WLST dispatching rule. In the case of instances with 100 activities, we generated 100 instances. In the case of instances with

TABLE 8. Summary of large problems for the proposed heuristic with the WLST dispatching rule.

# of activity	Project Makespan	Computation times (sec)	# of activity splitting
	no split	no split	
	with split	with split	
100 (Average)	69.5900	0.0494	12.3600
	60.2700	0.0583	
No split (Max)	114.0000	0.0624	–
Split (Max)	102.0000	0.0977	26.0000
200 (Average)	79.48	0.114	24.06
	66.32	0.129	
No split (Max)	113	0.1517	–
Split (Max)	92	0.1632	42
500 (Average)	92.00	0.587	59.68
	73.56	0.668	
No split (Max)	119	0.7128	–
Split (Max)	97	0.7887	120
1000 (Average)	108.5	2.184	127.48
	85.04	2.562	
No split (Max)	151	2.5754	–
Split (Max)	120	3.0869	225
2000 (Average)	125.36	8.413	247.86
	97.16	10.133	
No split (Max)	201	12.2751	–
Split (Max)	146	14.2894	438
5000 (Average)	154.6	61.044	654.8
	116.6	73.225	
No split (Max)	174	73.0324	–
Split (Max)	131	84.5205	901

5000 activities, the number of randomly generated instances is 5 due to long computation times for problem generation. The number of instances for other sizes (200, 500, 1000 and 2000) is 50.

These problems were generated with the same parameter settings used for smaller instances except for the number of activities. It can be seen in Table 8 that the computation times for solving large problems are reasonably good. For example when the number of activities is increased to 5000, the computation time is about a minute which is a reasonable and acceptable time for large problems [69].

5.5. Comparison with one of the most powerful algorithms and the best results available

In Section 2, we mentioned that the genetic algorithm of van Peteghem and Vanhoucke [64] was reported to give extremely good results for instances of sizes 30, 50 and 100. Their algorithm has been known as the most powerful heuristic developed until 2011 [73]. Papers such as [29, 70, 71] have been published after [64] but have not been able to give better results, as reported in [66]. An extensive version of the results of [64] is given in [67]. In this subsection, we compare our results to theirs for instances of sizes 30, 50 and 100. It is worth mentioning that in order to make our results comparable to theirs, resource vacation parameters are not considered in this section. This causes the makespans obtained through our method to be equal with and without splitting. We must also remind that we do not consider non-renewable resources in our method.

Their results were implemented for instances of size 30 of the project scheduling library PSPlib [46] (herein we may call them PSPlib30) and the Boctor multimode data set [17] instances which are of sizes 50 and 100. Later in reference [66], the same authors gave the results of [64] for the MMLIB dataset instances. Reference [64] is the only paper that solves the PSPlib30 instances both with and without non-renewable resources. Therefore, we can compare our results merely to them since we have no non-renewable resources. The PSPlib instances

TABLE 9. Comparison of our method to the one in [64] for PSPlib instances of size 30.

	[64]: % deviation from the lower bound by CPM		This paper: % deviation from the lower bound by CPM	
	Preemptive	Non- preemptive	Preemptive	Non- preemptive
Average	5.06%	5.60%	12.55%	12.55%
Max	76.00%	80.00%	120%	120%

TABLE 10. Comparison of our method to the one in [64] for Boctor instances of sizes 50 and 100.

# of activities		Complexity of the instances	[64]: % deviation from the lower bound by CPM	This paper: % deviation from the lower bound by CPM
Boctor50	Average	1.5885	21.52	47.3807
	Max	1.9231	47.66	79.4258
Boctor100	Average	1.6083	21.91	46.7502
	Max	1.8431	39.75	70.1266

TABLE 11. Summary of the properties of the PSPlib and Boctor datasets.

	PSPlib30			Boctor100		
	Average	Min	Max	Average	Min	Max
Order strength	0.46	0.34	0.61	0.87	0.79	0.93
Renewable resource strength	0.62	0.25	1	0.15	0.06	0.25
Renewable resource factor	0.75	0.5	1	0.88	0.67	1

have 3 modes for each activity and 2 renewable and 2 non-renewable resources. They can be downloaded from here [45]. The results of our comparison are given in Table 9. Note that the results are given for 552 out of 640 instances of the PSPlib which are feasible.

The network complexity for all PSPlib instances is 1.8125. As it can be seen, although we exhibited the superiority of our method to that of reference [21] in Section 5.3, the algorithm of [64] outperforms our method. However, our deviation percentages are not large in comparison to theirs, either. Therefore, the performance of our method is acceptable for PSPlib30. The ratios of our average deviations to theirs are 2.48 and 2.24 for the preemptive and non-preemptive cases, respectively.

We also compare our results to the results in [64] for the instances in the Boctor multimode library [17] which have 50 and 100 activities. This library consists of 120 instances for each of the sizes mentioned, which we call Boctor50 and Boctor100 herein. They have 1, 2, or 4 renewable resources, no non-renewable resources and 1, 2 or 4 modes for each activity [66]. The Boctor instances can be downloaded from [19]. The results are given in Table 10.

As seen in Table 10 for the Boctor dataset, the algorithm in [64] outperforms our method. The ratios of average deviations are 2.2 and 2.13 for the Boctor50 and Boctor100 datasets, respectively. However, it must be noted that the instances of the PSPlib and Boctor datasets are very limited in network complexity: as inferred from Table 10, none of their complexities are more than 2. Moreover, it is mentioned in [66] that in the dataset Boctor100, the average resource strength for each project is not larger than 0.25, which means that the resources are almost not restricted. The order strength of the projects is between 0.8 and 0.95, which means that the projects are mainly serial. The range for the values of the order strength (for PSPlib30 and Boctor100) is also rather limited. Moreover, most algorithms can solve the PSPlib instances to (near-) optimality[49,64]. Table 11 gives a summary of the properties of the PSPlib and Boctor datasets (Tab. 4 in [66]).

TABLE 12. Summary of the implementation of our method for the MMLIB.

# of activities		Complexity of the instances	MMLIB best results: % deviation from the lower bound by CPM	This paper: % deviation from the lower bound by CPM
MMLIB50	Average	3.9882	23.32	16.3032
	Max	5.2115	215.38	169.2308
MMLIB+ (first 1200 instances)	Average	6.4780	23.64	17.1808
	Max	7.9706	178.95	123.8095
	Average	3.9608	66.38	26.3967
	Max	5.2115	626.32	137.0370

TABLE 13. Multivariate regression for the Boctor and MMLIB+ instances.

Factors	# of resources	# of modes	Average resource strength	Network complexity	Intercept	R^2
t-statistics for Boctor50	5.018601*	−0.19011	−2.2063	−7.56495*	12.25214*	0.596022
t-statistics for Boctor100	5.187229*	5.487345*	−2.47922	−8.04583*	8.441775*	0.658656
t-statistics for MMLIB+	28.9001*	−6.14365*	−6.96471*	−19.3192*	18.93756*	0.541181

It is worth mentioning that the results given in Table 4 of [66] for the network complexity of Boctor100 are actually for Boctor50.

For the reasons mentioned, van Peteghem and Vanhoucke [66] created the MMLIB multimode library [67] which consists of three sets: MMLIB50, MMLIB100 and MMLIB+. The first is for instances of size 50 and the second is for size 100. Each of these libraries contains 540 instances of the sizes mentioned, all of which we tested. These libraries have 3 modes for each activity and 2 renewable and 2 non-renewable resources. The resource strengths and resource factors of both types of resources are set to 0.25, 0.50 or 0.75 and 0.50 or 1, respectively. MMLIB+ contains 3240 instances with 50 or 100 activities which have 3, 6 or 9 modes [50]. The instances have 2 or 4 renewable resources and 2 or 4 non-renewable ones. The resource strength of both resource types is set at 0.25, 0.50 or 0.75. In order to keep the number of instances reasonable, the resource factors of both resource types are set to 1. A feature of the MMLIB instances is that no modes can be excluded; therefore, the mode reduction procedure of [62] cannot be used as it was in reference [64]. This method reduces the number of modes by excluding the ones which are inefficient or non-executable. The authors in reference [62] define a mode to be inefficient if there is another mode of the same activity with equal or smaller duration and equal or less requirements of all resources. A mode is non-executable if it violates the resource constraints in any schedule. We chose the first 1200 instances of the MMLIB+ library to test our method on. More details and the best results for the MMLIB can be found in reference [67]. The results are given in Table 12.

Although the best known solutions for MMLIB are given for the case when both resource types are present[4], the deviations for our method are still small. Our results are particularly good for MMLIB+; the ratio of the average deviation of our method to that of the best results is much smaller for the MMLIB+ instances (0.3976 compared to 0.6991 and 0.7268 for MMLIB50 and MMLIB100, respectively).

In order to see what causes the solution qualities to be good, we carry out a multivariate regression similarly to what was done in [66]. We consider all the factors they did, except those related to non-renewable resources which we omit and the order strength which we replace by network complexity. We carry out this test for the MMLIB+ instances, like what they did. We also do this test for Boctor50 and Boctor100 as our worst results. The results are given in Table 13.

The factors which are significant for a t-test with at a 0.01 level of confidence are marked with asterisks. The R^2 values, or coefficients of determination, are between 0.54 and 0.66, which shows that the relationship between

[4]No results excluding the non-renewable resources exist as far as we have investigated.

TABLE 14. Parameters of standard instances.

Factors	Average # of resources	Average # of modes	Average resource strength (Average-Min-Max)	Average network complexity
Boctor50	2.33	2.536	0.113-0-0.25	1.5885
Boctor100	2.33	2.47	0.09-0.02-0.18	1.6083
MMLIB+ (first 1200 instances)	2.9	4.95 ≈ 5	0.635-0.204-7.93[6]	3.9608

the makespan and the factors is not very strong, as this was also the case in [66]. Table 13 shows that as the number of resources increases, the makespan also increases (since the t-statistic is positive). For the other three factors, however, the relation is reverse. Among these factors, the number of resources and network complexity (and also number of modes in Boctor100) are the significant factors for Boctor instances, while the first two have the greatest effects for the MMLIB+ solutions. A summary of the parameters of the standard instances needed for evaluating the regression results is given in Table 14[5] (resource strength values of each instance are averaged over all resources).

Now we give an analysis of Tables 13 and 14 in order to find out what caused the good results, especially for the MMLIB+ instances, in comparison to the Boctor instances. The resource strength parameter is only significant for the MMLIB+; therefore, it is excluded from our analysis. It can be seen for Table 13 that, with an increase in the number of resources, the performance of the algorithm decreases (the makespan increases). However, as seen in Table 14, the MMLIB+ − which exhibited better results − has a larger average number of resources compared to the other two datasets. Considering the large amount of t-statistic (28.9), this factor cannot be the reason for the good results of MMLIB+, since its effect is apparently overlapped by the effect of other factors, such as the network complexity which is significant and negative for all datasets. As we can see in Table 14, the average network complexity of the MMLIB+ is much larger than the same amount for the other two datasets, which can imply that our algorithm works better on instances with higher complexities. The number of modes is the last parameter, which is not significant for Boctor50 but is so for the other two datasets. However, its statistic is positive for Boctor100 and negative for MMLIB+; *i.e.*, the smaller number of modes, the smaller the makespan for Boctor100 and the larger the number of modes, the smaller the makespan for MMLIB+. We see in Table 14 that the average number of modes is small for Boctor100; however, we do not get good results, while this average is large for MMLIB+ and we get good results. The latter confirms that the number of modes can be another reason for the good solutions of the MMLIB+[7]. The intercept is significant for all datasets, which has a positive relationship with the makespan [66].

It must be noticed that the sensitivity of the solutions to the project parameters is the largest for MMLIB+, next for Boctor100, and then Boctor50 (except the intercept which is larger for Boctor50 than for Boctor100). Take the number of resources, for example: the statistic for the MMLIB+ is about 29, while it is about 5 for the Boctor datasets. This shows that with a certain amount of change in the number of resources, we should expect more changes in the MMLIB+ solutions than in the solutions of Boctor datasets.

Based on these results, we can infer that our heuristic method works well with instances of higher complexity and larger number of modes. This conclusion is based on comparison among the three datasets. For the MMLIB+ dataset considered solely, all of the factors indicated are significant.

It is worth mentioning that the best known solutions reported for MMLIB are slightly better than those reported by [64] for this dataset. For the MMLIB+ instances, the method in [64] is able to yield feasible solutions

[5]The differences between the amounts of resource strength reported in Table 11 from [66] may be due to two different methods of calculation for the parameter. For more information on this, see [65]. For the regression test (Tabs. 13 and 14), all the resource strength values are calculated with the same method.

[6]The large values of resource strength for some instances of MMLIB+ result from excess resource availabilities in those instances.

[7]We cannot say this in comparison to the results for Boctor100, since the average number of modes was small for Boctor100 but the results were bad. However, if MMLIB+ is considered solely, the average number of modes can be a reason for the good results we obtained since it is a significant parameter.

TABLE 15. Summary of the computation times for our method for standard datasets.

Dataset	Average computation time (No splitting − with splitting)	Maximal computation time (No splitting − with splitting)
PSPlib 30	0.025189 − 0.029991	0.0894 − 0.0919
Boctor50	0.12075 − 0.11851	0.18648 − 0.21567
Boctor100	0.24815 − 0.25333	0.37947 − 0.33894
MMLIB50	0.0217 − 0.02688	0.03262 − 0.04802
MMLIB100	0.04837 − 0.05327	0.06867 − 0.07672
MMLIB+ (first 1200 instances)	0.03304 − 0.03871	0.09254 − 0.10948

for 97.59% of the instances, although it should be emphasized that they consider non-renewable resources, too, which may justify this defect [66].

It is true that our algorithm was outperformed by [64] for the Boctor dataset, and did not yield very good results for the PSPlib30 dataset in comparison to that paper. However, the small average deviations obtained by our heuristic for the MMLIB instances imply that our method has an acceptable performance for tested instances with higher complexities. Therefore, our method can yield acceptable results, since its performance was bad, acceptable and good for the Boctor, PSPlib30 and MMLIB instances, respectively. Our results are especially good for the MMLIB+ instances.

The solution times of [64] are very reasonable −0.05, 0.24 and 2.46 s for size 30, for 1000, 5000 and 50 000 schedules generated, respectively (these times are for the presence of non-renewable resources. The solution times for the other case are not given). Our solution times are given in Table 15.

The solution times for the Boctor dataset are the largest among others. In essence, we can infer that our heuristic method performs extremely well in comparison to [21], but does not work very well in comparison with [64] for instances of low complexity. Instead, it works well for tested instances of high complexity regarding their best known solutions. Our results are especially good for the MMLIB+ instances. The best known solutions mentioned are slightly better than the results of [64] for the MMLIB instances; sometimes, their method fails to find feasible solutions for the MMLIB+ instances, as mentioned. From Sections 5.3−5.5, we may say that the greater the number of activities and the network complexity, the better our heuristic performs. This quality may be useful in real-world scheduling problems.

5.6. The drawbacks of presetting the modes

In Section 5.2, we investigated our heuristic regardless of the mode selection method by solving the RCPSP obtained through using (4.4) in GAMS. In this section, we verify the accuracy of our mode selection method independently of that of our heuristic.

Characteristics of a good mode vector are time feasibility, short length of the critical path in the associated project and resource feasibility [15]. A good (near-optimal and near-feasible) mode vector selected before applying the heuristic can make the heuristic method yield a good solution. In order to evaluate our mode selection method, we consider the deviation of the project length associated with the selected modes (calculated via the CPM) from the lower bound of the project. In terms of feasibility, we consider the percentage of total resource requirements that cannot be satisfied to the total resource requirements, when activities are scheduled according to the selected mode vector at their earliest start times.

We propose another scheme for choosing the optimal mode vector using the notion of the SFM rule [18]. We do this to show that a better mode vector for the first phase can yield better schedules for the MR-CPSP. The pseudo-code of this mode selection method is presented in Figure 2 (besides the notations of the model (3.1)−(3.10), (4.1)−(4.4) and the heuristic procedure presented in Section 4.3, consider the following: $\mathrm{Pred}(j)$ is the set of predecessors of $j = 1, \ldots, J$ and $\tilde{K}_r(t) = K_{rt} - \sum_{j \text{ is being executed in } t} k_{jmr}$ is the remaining amount of resource r availability at time t, $r = 1, \ldots, R, t = 1, \ldots, T$ − the notations and presentation of the

algorithm are inspired by the one in [21]):

Initialization: $d_j = Sort(d_j), \forall j = 1, \ldots, J$ in ascending manner
For $j = 1, \ldots, J$,
  If $\exists m', m'' : d_{jm'} = d_{jm''}$
    If $\sum_{r=1}^{R} k_{jm'\,r}^{NWR(r)} < \sum_{r=1}^{R} k_{jm''\,r}^{NWR(r)}$
      Choose m' as the mode with the smaller duration
    Else
      Choose m'' as the mode with the smaller duration
    End
  End
End

$ES_1 = 0$

For $j = 2$ to J
  For $i \in \text{Pred}\,(j)$
    If m_i is already set
      Continue
    End
    For $u = 1$ to M_i
      Key $= 0$
      For $t = ES_i$ to $ES_i + d_{iu}$
        If $\exists r \in 1, \ldots, R : k_{iur} > \tilde{K}_r(t)$
          Key $= 1$
          Break
        End
      End
      If Key $= 0$
        $m_i = u$
        Break
      End
    End
    If m_i is not chosen yet
      $m_i = \arg\min_u \left\{ \sum_{r=1}^{R} k_{jur}^{NWR(r)} \right\}$
    End
    For $t = ES_i$ to $ES_i + d_{im_i}$
      $\tilde{K}_r(t) = \tilde{K}_r(t) - k_{im_i r}, \forall r \in 1, \ldots, R$
    End
    $EF_i = ES_i + d_{im_i}$
  End
  $ES_j = \max_i \{EF_i\}$
End

Figure 2. Another scheme for choosing the optimal mode vector using the notion of the SFM rule.

In step 3, the rule for choosing among the modes when all of them are infeasible is similar to the LCR (Least Criticality Ratio) and LRP (Least Resource Proportion) rules (for more information, see [18]) We implement

TABLE 16. Summary of the evaluation of the mode selection methods for generated instances.

# of activity	Obj function (no splitting − with splitting)	% Deviation of 1st phase from LB	% Resource infeasibility of 1st phase	Obj function (no splitting − with splitting)	% Deviation of 1st phase from LB	% Resource infeasibility of 1st phase
30	48.5300 − 44.9820	5.1171	22.3387	72.5960 − 60.8540	51.6313	17.7053
Max	85.0000 − 85.0000	35.2941	47.6117	135.0000 − 124.0000	166.6667	42.6843
60	65.0800 − 59.3900	5.6313	22.3959	96.9050 − 73.4600	43.9765	18.1591
Max	102.0000 − 101.0000	26.1905	42.6036	156.0000 − 128.0000	142.4242	38.5542
90	68.0300 − 58.5800	5.7394	23.5645	103.8100 − 76.4750	42.3993	18.8854
Max	115.0000 − 97.0000	21.8750	42.9660	164.0000 − 145.0000	94.1176	35.2970

TABLE 17. Summary of the evaluation of the first mode selection method for standard instances.

Dataset	Obj function	% deviation of 1st phase from LB (average − maximum)	% resource infeasibility of 1st phase (average − maximum)
PSPlib 30	33.49819 − 63	1.53733 − 20	7.215126 − 40.54054
Boctor50	337.4 − 433	8.9598 − 36.9565	18.7617 − 27.2982
Boctor100	663.9333 − 857	7.9438 − 26.1450	18.8897 − 27.5140
MMLIB50	34.0111 − 75	2.5743 − 25	11.3000 − 57.5198
MMLIB100	42.7593 − 100	2.3256 − 21.0526	12.7209 − 54.5499
MMLIB+ (first 1200 instances)	61.6758 − 231	0.2368 − 19.0476	16.5340 − 53.6810

both mode selection methods for the WLST dispatching rule in sets of instances with 30, 60 and 90 activities, generated by the modified ProGen, for the parameter settings of Tables 1-3. 500 instances of size 30 and 200 instances of sizes 60 and 90 were evaluated. The results are shown in Table 16.

As it is seen in Table 16, the first mode selection method has much lower deviation percentages, and infeasibility percentages very close to those of the second method. Moreover, the makespans obtained using the first method are much better than the ones obtained using the second; this shows that, as mentioned, better (in terms of deviation and resource infeasibility) mode vectors selected in the first phase lead to better makespans for the MRCPSP. At a cost of only a few percentages of infeasibility, the first method gives drastically smaller deviations from the lower bound for the first phase. Its results are also acceptable for a first phase solution in terms of resource infeasibility. Interesting results are also given for standard instances in Table 17.

As it can be seen, our first mode selection method again yields very good results both in terms of deviation from the lower bound and resource feasibility, especially for the PSPlib and MMLIB instances (for which our algorithm had an acceptable and good performance, respectively − see Sect. 5.5). These small deviation amounts such as 1.53733 for the PSPlib and 0.2368 for the MMLIB+ show that the chosen modes are very close to smallest duration modes − similarly to the notion of the SFM rules.

Finally, it can be mentioned that predetermining the modes in the MRCPSP can, while reducing complexity, help determine good-quality answers for the problem; just as mentioned for the MRCPSP/Max in [7]. These results give rise to speculations such as whether the complexity arising from adding multiple modes to the RCPSP is necessary at all, and whether the modes can be selected previously.

6. CONCLUDING REMARKS

We presented a new heuristic solution procedure for multi-mode resource-constrained project scheduling problems which allows activities to be split due to resource vacations. A method was also devised for selecting

one execution mode among several modes for each activity in order to reduce the problem complexity. The computational results showed that the proposed heuristic method significantly reduces the computational time. It also improves the project makespan as compared with the existing ones in reference [21], especially as the size of the problem grows larger.

We also validated the mode selection method by calculating the amounts of deviation from the lower bound of the makespan and also resource infeasibility of the path associated with the selected mode vector. The results showed that our mode selection method performs well. Our results also indicated that better mode vectors selected in the first phase lead to better makespans for the MRCPSP. These results, along with those of similar two-phase methods for the MRCPSP/Max, give rise to speculations such as whether the complexity imposed on the problem by adding multiple modes to the RCPSP is necessary at all, and whether the modes can be selected previously.

In order to have a base for evaluating our heuristic method, we also generated some instances with the same parameters used for our heuristic and the competing heuristics, and tried to solve them with GAMS commercial solver. Our heuristic significantly reduced the computational time in comparison with this solver We also corrected some erroneous constraints in the mathematical model of the problem. Moreover, in order to justify the use of our heuristic method, we also solved the second phase of our method with the commercial software GAMS using the mode vector chosen in the first phase. It was shown that, even for medium-sized instances of 60 activities, it was extremely hard and time-consuming to solve, at least when resource vacations were present.

In essence, we can infer that our heuristic method performs very well in comparison to [21], but does not work very well in comparison with [64] for tested instances of low complexity. Instead, it works well for tested instances of higher complexity. Overall, we may conclude that the greater the number of activities and the network complexity, the better our heuristic may perform. This quality may be useful in real-world scheduling problems.

An avenue for extension of this work is implementing and doing convergence analysis for an algorithm which re-solves the mixed integer optimization model to find the optimal modes with the order in the schedule found using the heuristic.

Conflict of interest

The authors declare that they have no conflict of interest.

Acknowledgements. The authors thank the anonymous referees for their useful comments.

References

[1] B. Afshar-Nadjafi, H. Najjarbashi and E. Mehdizadeh, A branch-and-bound procedure for resource leveling in multi-mode resource constraint project scheduling problem. *Res. J. Recent Sci.* **1** (2012) 33–38.

[2] P. Agrawal and S. Rao, Energy-aware scheduling of distributed systems. *IEEE Trans. Autom. Sci. Eng.* **11** (2014) 1163–1175.

[3] J. Alcaraz, C. Maroto and R. Ruiz, Solving the multi-mode resource constrained project scheduling problem with genetic algorithms. *J. Oper. Res. Soc.* **54** (2003) 614–626.

[4] N. Anne and V. Muthukumar, Energy aware scheduling of aperiodic real-time tasks on multiprocessor systems. *J. Comput. Sci. Eng.* **7** (2013) 30–43.

[5] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.* **53** (2004) 584–600.

[6] F. Ballestın, A. Barrios and V. Valls, An evolutionary algorithm for the resource-constrained project scheduling problem with minimum and maximum time-lags. *J. Scheduling* **14** (2009) 391–406.

[7] F. Ballestin, A. Barrios and V. Valls, Looking for the best modes helps solving the MRCPSP/max. *Int. J. Prod. Res.* **51** (2013) 813–827.

[8] E. Bampis, V. Chau, D. Letsios, G. Lucarelli, I. Milis and G. Zois, Energy efficient scheduling of MapReduce jobs, in *Proc. of Euro-Par 2014: Parallel Processing: 20th International Conference.* Edited by F. Silva, I. Dutra and V. Santos Costa. Springer, Porto (2014) 198–209.

[9] N. Bansal and K. Pruhs, The geometry of scheduling, in *Proc. of FOCS'10, The IEEE 51st Annual Symposium on Foundations of Computer Science.* IEEE, Washington DC, USA (2010) 407−414.

[10] N. Bansal, H.L. Chan, R. Khandekar, K. Pruhs, B. Schieber and C. Stein, Non-preemptive min-sum scheduling with resource augmentation, in *Proc. of 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS'07*. IEEE, Providence, RI (2007) 614–624.

[11] N. Bansal, T. Kimbrel and K. Pruhs, Speed scaling to manage energy and temperature. *J. ACM* **54** (2007) Article No. 3.

[12] N. Bansal, K. Pruhs and C. Stein, Speed scaling for weighted flow time, in *Proc. of SODA'07, The Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, USA (2007) 805–813.

[13] N. Bansal, H.L. Chan, K. Pruhs and D. Katz, Improved bounds for speed scaling in devices obeying the cube-root rule. In *Automata, languages and programming*. Springer, Berlin Heidelberg (2009) 144–155.

[14] N. Bansal, H.L. Chan and K. Pruhs, Speed scaling with an arbitrary power function. *ACM Trans. Algorithms* **9** (2013) Article No. 18.

[15] A. Barrios, F. Ballestin and V. Valls, A double genetic algorithm for the MRCPSP/max. *Comput. Oper. Res.* **38** (2011) 33–43.

[16] T. Berthold, S. Heinz, M.E. Lubbecke, R.H. Mohring and J. Schulz, A constraint integer programming approach for resource-constrained project scheduling. In Integration of AI And OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer, Berlin Heidelberg (2009) 313–317.

[17] F.F. Boctor, Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *Int. J. Prod. Res.* **31** (1993) 2547–2558.

[18] F.F. Boctor, A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. *Eur. J. Oper. Res.* **90** (1996) 349–361.

[19] F.F. Boctor, Other benchmarks (2004) Available at: http://www.om-db.wi.tum.de/psplib/dataob.html. Accessed 10 April 2015.

[20] K. Bouleimen and H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *Eur. J. Oper. Res.* **149** (2003) 268–281.

[21] J. Buddhakulsomsiri and D.S. Kim, Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. *Eur. J. Oper. Res.* **175** (2006) 279–295.

[22] R.A. Carrasco, *Resource Cost Aware Scheduling*. Ph.D. thesis, Columbia University, Canada (2013).

[23] R.A. Carrasco, G. Iyengar and C. Stein, Energy aware scheduling for weighted completion time and weighted tardiness. Technical report. Preprint arXiv:1110.0685 (2011).

[24] R.A. Carrasco, G. Iyengar and C. Stein, Single machine scheduling with job-dependent convex cost and arbitrary precedence constraints. *Oper. Res. Lett.* **41** (2013) 436–441.

[25] S. Chakrabarti, C.A. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein and J. Wein, Improved scheduling algorithms for minsum criteria (extended abstract). In *Proc. of Automata, Languages and Programming: 23rd International Colloquium, ICALP '96, Paderborn, Germany, July 8-12, 1996*. Springer Science & Business Media (1996) 646−657.

[26] J. Cheng, J. Fowler, K. Kempf and S. Mason, Multi-mode resource-constrained project scheduling Problems with nonpreemptive activity splitting. *Comput. Oper. Res.* **53** (2015) 275–287.

[27] T.C.E. Cheng, A. Janiak and M.Y. Kovalyov, Bicriterion single machine scheduling with resource dependent processing times. *SIAM J. Optim.* **8** (1998) 617–630.

[28] T.C.E. Cheng, Q. Ding and B.M.T. Lin, A concise survey of scheduling with time-dependent processing times. *Eur. J. Oper. Res.* **152** (2004) 1–13.

[29] J. Coelho and M. Vanhoucke, Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *Eur. J. Oper. Res.* **213** (2011) 73–82.

[30] A. Drexl and J. Gruenewald, Nonpreemptive multi-mode resource constrained Project scheduling. *IIE. Trans.* **25** (1993) 74–81.

[31] I. Goiri, F. Julià, R. Nou, J.L. Berral, J. Guitart and J. Torres, Energy-aware scheduling in virtualized datacenters, in *Proc. of the 2010 IEEE International Conference on Cluster Computing*. IEEE, Heraklion, Crete (2010) 58–67.

[32] A. Grigoriev, M. Sviridenko and M. Uetz, Machine scheduling with resource dependent processing times. *Math. Program.* **110** (2007) 209–228.

[33] L.A. Hall, D.B. Shmoys and J. Wein, Scheduling to minimize average completion time: off-line and on-line algorithms, in *Proc. of ACM-SIAM Symposium on Discrete Algorithms, SODA 7*. New Orleans, Louisiana (1996) 142–151.

[34] L.A. Hall, A.S. Schulz, D.B. Shmoys and J. Wein, Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Math. Oper. Res.* **22** (1997) 513–544.

[35] S. Hartmann, Project scheduling with multiple modes: a genetic algorithm. *Ann. Oper. Res.* **102** (2001) 111-135.

[36] S. Hartmann and D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **207** (2010) 1–14.

[37] R. Heilmann, Resource–constrained project scheduling: a heuristic for the multi-mode case. *OR. Spektrum.* **23** (2001) 335–357.

[38] R. Heilmann, A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *Eur. J. Oper. Res.* **144** (2003) 348–365.

[39] A. Janiak and M.Y. Kovalyov, Single machine scheduling subject to deadlines and resource dependent processing times. *Eur. J. Oper. Res.* **94** (1996) 284–291.

[40] B. Jarboui, N. Damak, P. Siarry and A. Rebai, A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Appl. Math. Comput.* **198** (2008) 299–308.

[41] J. Josefowska, M. Mika, R. Rozycki, G. Waligora and J. Weglarz, Simmulated annealing for multi-mode resource constrained project scheduling. *Ann. Oper. Res.* **102** (2001) 137–155.

[42] B. Kalyanasundaram and K. Pruhs, Speed is as powerful as clairvoyance. *J. ACM.* **47** (2000) 617–643.
[43] A. Kandhalu, J. Kim, K. Lakshmanan and R. Rajkumar, Energy-aware partitioned fixed-priority scheduling for chip multiprocessors, in *Proc. 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.* IEEE, Toyama (2011) 93–102.
[44] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. *Eur. J. Oper. Res.* **90** (1996) 320–333.
[45] R. Kolisch and A. Sprecher, Multi Mode Data Sets (1996). Available at http://www.om-db.wi.tum.de/psplib/datamm.html. Accessed 10 April 2015.
[46] R. Kolisch and A. Sprecher, PSPLIB a project scheduling problem library. *Eur. J. Oper. Res.* **96** (1997) 205–216.
[47] R. Kolisch, A. Sprecher and A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems. *Manag. Sci.* **41** (1995) 1693–1703.
[48] A. Lova, P. Tormos and F. Barber, Multi-mode resource constrained project scheduling: scheduling schemes, priority rules and mode selection rules. *Intel. Artif.* **30** (2006) 69–86.
[49] A. Mastor, An experimental and comparative evaluation of production line balancing techniques. *Manag. Sci.* **16** (1970) 728–746.
[50] T. Messelis and P. De Causmaecker, An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **233** (2014) 511–528.
[51] G.D. Micheli, Synthesis and Optimization of Digital Circuits. McGraw-Hill (1994).
[52] R. Mishra, N. Rastogi, D. Zhu, D. Mossé and R. Melhem, R., Energy aware scheduling for distributed real-time systems, in *Proc. of the International Parallel and Distributed Processing Symposium.* IEEE Computer Society (Washington), Nice, France (2003) 21–29.
[53] K.T. Nguyen, Lagrangian Duality in Online Scheduling with Resource Augmentation and Speed Scaling. In Algorithms – ESA 2013. Vol. 8125 of *Int. Lect. Notes Comput. Sci.* Springer, Berlin Heidelberg (2013) 755–766.
[54] C.A. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein and J. Wein, Improved bounds on relaxations of a parallel machine scheduling problem. *J. Comb. Optim.* **1** (1998) 413–426.
[55] C. Phillips, C. Stein and J. Wein, Minimizing average completion time in the presence of release dates. *Math. Program.* **82** (1998) 199–223.
[56] C. Phillips, C. Stein, E. Torng and J. Wein, Optimal time-critical scheduling via resource augmentation. *Algorithmica* **32** (2002) 163–200.
[57] J. Polo, C. Castillo, D. Carrera, Y.W.I. Becerra, M. Steinder, J. Torres and E. Ayguade, Resource-Aware Adaptive Scheduling for MapReduce Clusters, in *Proc. of Middleware'11 of the 12th ACM/IFIP/USENIX international conference on Middleware*, edited by F. Kon and A.M. Kermarrec. Springer-Verlag, Lisboa (2011) 187–207.
[58] M. Seyed-Hosseini and S.M. Sabzehparvar, A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. *J. Supercomput.* **44** (2008) 257–273.
[59] N. Sharma, V. Sahula and C.P. Ravikumar, Energy aware task scheduling for soft real time systems using an analytical approach for energy estimation. *Int. J. Comput. Sci. Eng.* **1** (2012) 33–39.
[60] H. Simonis and T. Hadzic, A Resource Cost Aware Cumulative. In *Recent Advances in Constraints: 14th Annual ERCIM International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2009, Barcelona, Spain, June 15-17, 2009,* Revised Selected Papers. Springer-Verlag, Berlin Heidelberg (2011) 76–89.
[61] A. Sprecher and A. Drexl, Solving Multi-mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part I: Theory. Working paper (1996).
[62] A. Sprecher, S. Hartmann and A. Drexl, An exact algorithm for the project scheduling with multiple modes. *OR. Spektrum.* **19** (1997) 195–203.
[63] M. Tillenius, E. Larsson, R.M. Badia and X. Martorell, Resource-aware task scheduling. *ACM Trans. Embed. Comput. Syst.* **14** (2015) Article no. 5.
[64] V. van Peteghem and M. Vanhoucke, A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **201** (2010) 409–418.
[65] V. van Peteghem and M. Vanhoucke, Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. *J. Heuristics* **17** (2011) 705–728.
[66] V. van Peteghem and M. Vanhoucke, An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *Eur. J. Oper. Res.* **235** (2014) 62–72.
[67] V. van Peteghem and M. Vanhoucke, Multiple Modes |Operations Research & Scheduling research Group (2011) Available http://www.projectmanagement.ugent.be/?q=research/project_scheduling/multi_mode. Accessed 10 April 2015.
[68] S. Viswanathan, B. Veeravalli and T.G. Robertazzi, Resource-aware distributed scheduling strategies for large-scale computational cluster/grid systems. *IEEE Trans. Parallel Distrib.* **18** (2007) 1450–1461.
[69] S. Vob and A. Witt, Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. *Int. J. Prod. Econ.* **105** (2007) 445–458.
[70] L. Wang and C. Fang, An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem. *Comput. Oper. Res.* **39** (2012) 449–460.
[71] T. Wauters, K. Verbeeck, G. Vanden Berghe and P. De Causmaecker, Learning agents for the multi-mode project scheduling problem. *J. Oper. Res. Soc.* **62** (2011) 281–290.
[72] J. Wêglarz, Project scheduling with continuously-divisible, doubly constrained resources. *Manag. Sci.* **27** (1981) 1040−1053.

[73] J. Weglarz, J. Józefowska, M. Mika and G. Waligora, Project scheduling with finite or infinite number of activity processing modes a Survey. *Eur. J. Oper. Res.* **208** (2011) 177–205.

[74] K.K. Yang, A comparison of dispatching rules for executing a resource-constrained project. *Omega* **26** (1998) 729–738.

[75] M. Yong, N. Garegrat and S. Mohan, Towards a Resource Aware Scheduler in Hadoop, in *Proc. of ICWS 2009*. IEEE, Los Angeles (2009) 102–109.

[76] B.D. Young, S. Pasricha, A.A. Maciejewski, H.J. Siegel and J.T. Smith, Heterogeneous Makespan and Energy-Constrained DAG Scheduling, in *Proc. of Workshop on Energy Efficient High Performance Parallel and Distributed Computing*. ACM, New York, New York (2013) 3–12.

[77] Q. Zhang, M.F. Zhani, Y. Yang, R. Boutaba and B. Wong, PRISM: fine-grained resource-aware scheduling for MapReduce. *IEEE Trans. Cloud. Comput.* **3** (2015) 182–194.

[78] G. Zhu, J.F. Bard and G. Yu, A branch-and-cut procedure for the multimode resource-constrained project scheduling problem. *Informs J. Comput.* **18** (2006) 377–390.