J Sched
DOI 10.1007/s10951-015-0422-4

An integer programming approach to the multimode resource-constrained multiproject scheduling problem

Túlio A. M. Toffolo · Haroldo G. Santos · Marco A. M. Carvalho · Janniele A. Soares

Abstract The project scheduling problem (PSP) is the subject of several studies in computer science, mathematics, and operations research because of the hardness of solving it and its practical importance. This work tackles an extended version of the problem known as the multimode resource-constrained multiproject scheduling problem. A solution to this problem consists of a schedule of jobs from various projects, so that the job allocations do not exceed the stipulated limits of renewable and nonrenewable resources. To accomplish this, a set of execution modes for the jobs must be chosen, as the jobs' duration and amount of needed resources vary depending on the mode selected. Finally, the schedule must also consider precedence constraints between jobs. This work proposes heuristic methods based on integer programming to solve the PSP considered in the *Multidisciplinary International Scheduling Conference: Theory and Applications* (MISTA) 2013 Challenge. The developed solver was ranked third in the competition, being able to find feasible and competitive solutions for all instances and improving best known solutions for some problems.

Keywords Multimode resource-constrained multiproject scheduling problem · Integer programming · Matheuristics · MISTA Challenge

T. A. M. Toffolo (✉) · H. G. Santos · M. A. M. Carvalho
Computing Department, Federal University of Ouro Preto, Ouro Preto, Brazil
e-mail: tulio@toffolo.com.br

H. G. Santos
e-mail: haroldo@iceb.ufop.br

M. A. M. Carvalho
e-mail: mamc@iceb.ufop.br

T. A. M. Toffolo
CODeS, Department of Computer Science, KU Leuven, Ghent, Belgium

J. A. Soares
Information Systems Department, Federal University of Ouro Preto, Ouro Preto, Brazil
e-mail: janniele@decsi.ufop.br

1 Introduction

A *project scheduling problem* (PSP), in its general form, consists in scheduling the processing times of *jobs* (or activities) contained in a project. These jobs are interrelated by precedence constraints, that is, a job may require another job to be finished before its start. This class of problems models many situations of practical interest in engineering and management sciences in general and has been tackled by experts in various fields ranging from civil engineering in De Marco (2011) to software development in Alba and Chicano (2007). One natural extension of the PSP is the *resource-constrained project scheduling problem* (RCPSP), which adds constraints on different resources consumed during the processing of each job. The RCPSP is NP-hard in the strong sense (Blazewicz et al. 1983) and was claimed to be "one of the most intractable problems in operations research" by Möhring et al. (2003).

Various formulations to the RCPSP based on mixed integer linear programming (MILP) are found in the literature. Pritsker et al. (1969) proposed a binary programming formulation where the variables x_{jt} indicate whether a job j is scheduled at time t ($x_{jt} = 1$). In this formulation, known as a discrete-time formulation, the number of binary decision variables depends on an upper limit $\bar{t}$ for the number of timeslots required to complete the project, which can be heuristically defined. Thus, the number of variables in this formulation is $O(n \times \bar{t})$. Kolisch and Hartmann (2006) extended this formulation to handle different execution modes, adding an additional index to the binary variables. Kone et al. (2011) proposed a formulation based on events called on/off event-based (OOE). An event corresponds to the starting time of one or more jobs. General integer variables t_e are used to indicate the starting time of each event e. These variables are linked to binary variables x_{je} that indicate whether task j is starting or is still being processed at event e. Because the maximum number of events is clearly equal to the number of jobs n, this formulation has $O(n^2)$ variables. The number of constraints in this formulation is $O(n^3)$, making it rather impractical for use in standalone MIP solvers.

There are many other extensions of the PSP and RCPSP. For in-depth research on PSPs' historical origins, classification, complexity analysis, and solution methods, we refer the reader to Weglarz (1999); Klein (2000); Demeulemeester and Herroelen (2002); Józefowska and Weglarz (2006); Hartmann (2002) and Artigues et al. (2013). Additionally, a comprehensive classification and computational analysis of heuristics and metaheuristics applied to the RCPSP is provided in Kolisch and Hartmann (1999), and an updated version is found in Kolisch and Hartmann (2006).

The PSP considered in the paper is a more recent RCPSP generalization: the *multimode resource-constrained multiproject scheduling problem* (MRCMPSP). This problem was chosen to be the subject of the *Multidisciplinary International Scheduling Conference: Theory and Applications* (MISTA) Challenge 2013 (Wauters et al. 2014). Our approach, a hybrid solver based on integer programming (IP) components, won third place in this challenge.

The MRCMPSP is stated as follows: a set P of projects, each project $p \in P$ consisting of a set $J_p = \{1, \ldots, |J_p|\}$ of nonpreemptive jobs, must be scheduled. Each project p also has a *release time*, that is, a time when the processing of its jobs may be started. The start and end of a project are delimited by *dummy* jobs 0 and $|J_p|+1$, respectively the first and last jobs of each project.

To schedule a project means to determine the starting time of all of its jobs, subject to the precedence constraints among them and their *resource* consumption given the available resources. Jobs may consume *local* resources – exclusive resources of a project – and *global resources* – resources shared among all projects. These resources can be either *renewable* – with capacity fixed per time unit during the project duration – or *nonrenewable* – with capacity fixed per project duration. Any given job may be executed in one or more *execution modes*, each requiring a specific amount of resource consumption and resulting in different durations for a job's completion. Note that dummy jobs have no resource consumption and their duration is always zero.

One practical example of a problem that can be modeled as the MRCMPSP is software project management. Project management is a key activity for software projects. In Kerzner (2009) it is stated that one of the most important responsibilities of a project manager is to plan the integration and execution of activities. Project managers often prioritize the control of resources and require a detailed and formal plan. The application of optimization techniques to problems of software engineering has attracted growing interest in the academic community, creating a new field called search-based software engineering. In the work of Alba and Chicano (2007), for instance, the resources are the employees, each with a set of skills and wages, plus a maximum degree of dedication to the project. Genetic algorithms are used to find good solutions, minimizing the cost and time of software projects.

Some methods were recently proposed for the MRCMPSP. Asta et al. (2014), winners of the MISTA Challenge, proposed an approach combining Monte Carlo and hyperheuristics, along with several neighborhoods, which are explored by stochastic local searches. This approach uses an indirect solution representation where solutions are always decoded by a constructive algorithm that generates feasible solutions w.r.t. precedences and renewable resource usage. To speed up the constructive algorithm, a prefix matching method is employed. In the construction phase, modes are randomly selected and the feasibility of using nonrenewable resources is achieved by a stochastic local search. Several neighborhoods that perform large modifications in the solution were developed, such as a neighborhood where the allocation priorities of all tasks of a given project are changed.

Geiger (2013) proposed an iterative variable neighborhood search for the MRCMPSP. His local search method explores the solution space through systematic exchanges of neighborhood structures (Hansen and Mladenović 1997). In this approach, a set of feasible schedules X associated with two vectors, $M = \{m_1, \ldots, m_n\}$ and $S = \{S_1, \ldots, S_n\}$, is presented. M represents the execution mode chosen for each job, and S is the permutation of job indexes. To generate the initial feasible schedule, this approach randomly assigns modes to the vector M. If M is not feasible regarding nonrenewable resources, a procedure that randomly changes modes is applied until the feasibility of a maximum number of iterations is reached. If feasibility is not obtained, M is rebuilt. Subsequently the sequence S is built, assigning higher priority to activities with shorter starting times. The local search is performed in parallel. Once a local optimum is reached, the best solution found so far is updated and the search continues with the best known alternative.

A lower bound on a project's earliest finish time is the *critical path duration*. The *critical path method* (Kelley Jr and Walker 1959) is a tool for general project management that represents the precedence constraints as a network, where each job is a node and arcs connect jobs to their successors and predecessors. This method computes the earliest and latest start and finish times for each job such that the project is not delayed, while precedence constraints are observed. The critical path itself is the sequence of related jobs that cannot be delayed without delaying the whole project, denoted by a path between the two dummy jobs in the network. The critical path duration is the sum of these jobs' durations. To calculate the critical path duration for a MRCMPSP instance, some constraints are ignored: the duration of a job is fixed as the duration of the fastest execution mode and all resource constraints are ignored. In the MRCMPSP, once a project is scheduled, it is assigned a *makespan*, defined as the difference between the project finish and release times, and a *project delay*, defined as the difference between the project critical path duration and the actual project duration.

To measure the quality of the solutions submitted to the MISTA Challenge, an objective function with two components was proposed: to minimize the *total project delay* (TPD) and the *total makespan* (TMS). The TPD is defined as the sum of all project delays, and the TMS is defined as the time required to finish all projects, i.e., the difference between the maximum finish time of a project and the minimum release time of a project. The first component is the main objective, while the second is a tiebreaker.

To address the MRCMPSP, we produced a matheuristic (Maniezzo et al. 2010), i.e., a hybrid algorithm with several IP-based components. These algorithms are detailed in the following sections.

The remainder of this paper is organized as follows. The next section presents an IP formulation for the MRCMPSP. Section 3 presents the general idea of the proposed algorithm. Section 4 describes the IP heuristic decomposition used to generate the initial solutions, and Sect. 5 details the hybrid local search. Section 6 reports the results obtained with the formulation and with the proposed method for the MISTA 2013 Challenge instances. Finally, conclusions are drawn in Sect. 7.

2 An integer programming formulation for the MRCMPSP

We present in this section an IP formulation for the problem based on the formulations presented by Kolisch and Sprecher (1997) and Pritsker et al. (1969). These authors proposed time-indexed formulations for the RCPSP, meaning that the number of variables increases with the duration of jobs and projects. Kone et al. (2011) proposed an event-based formulation that does not depend on the duration of jobs and projects. The number of events is given by the total number of jobs. Thus, their formulation is specially interesting for instances with long-duration jobs, i.e., for instances in which the number of jobs is smaller than the number of time slots. Kone et al. (2011) presented situations in which the event-based formulation outperforms the time-indexed formulation. In our experiments on the MISTA 2013 Challenge instances, however, the event-based formulation performed poorly, being unable to solve some of the easy instances using commercial solvers. Therefore, we focused on the time-indexed models.

To present the integer formulation for the MRCMPSP, the following input data are considered:

P:	set of projects;
J:	set of jobs;
J_p:	set of jobs on project p such that $J_p \subseteq J \; \forall p \in P$;
M_j:	set of modes for job j;
K:	set of nonrenewable resources;
R:	set of renewable resources;
T:	set of available timeslots $\{1, ..., \lvert T \rvert\}$;
T_{jm}:	set of possible timeslots to start job j on mode m, $\{te_j, ..., tl_{jm}\}$, where te_j is the earliest starting time of job j given by the critical path and tl_{jm} is the latest starting time of job j in mode m such that the job ends within $\lvert T \rvert$ timeslots – one could set, for instance, $tl_{jm} = \lvert T \rvert - d_{jm}$;
B:	set of precedence relations expressed by ordered pairs (j, l), where j should be scheduled before l;
B_j:	set of immediate predecessors of job j;
d_{jm}:	duration of job j in mode m;
u_{kjm}:	required units of nonrenewable resource k for job j to be processed in mode m;
v_{rjm}:	required units of renewable resource r for job j to be processed in mode m;
o_k:	available units of nonrenewable resource k;
q_r:	available units of renewable resource r;
ω_1:	weight in objective function for delaying a project;
ω_2:	weight in objective function for the TMS.

Note that the number of timeslots $\lvert T \rvert$ must be large enough to allow the production of feasible schedules.

The following binary decision variables are used to select the mode and starting time for jobs:

x_{jmt}:	binary variable that is equal to 1 when job j is processed in mode m and starts at time t, and 0 otherwise;

Additionally, we employ the following auxiliary general integer variables:

y_p:	completion time for project p;
z:	completion time for latest project.

The objective function minimizes the sum of completion times for projects as well as the completion time of the last project.[1]

[1] Constants included in the objective function presented in the MISTA Challenge problem description were omitted for the sake of clarity.

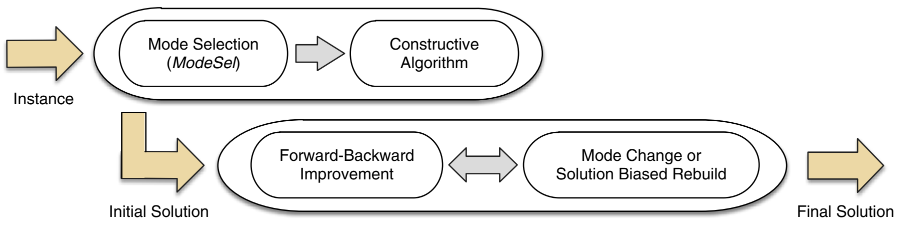
Instance
Mode Selection (*ModeSel*)
Constructive Algorithm
Initial Solution
Forward-Backward Improvement
Mode Change or Solution Biased Rebuild
Final Solution

Fig. 1 Outline of developed solver

Minimize

$$\omega_1 \sum_{p \in P} y_p + \omega_2 z \tag{1}$$

subject to

$$\sum_{m \in M_j} \sum_{t \in T_{jm}} x_{jmt} = 1 \forall j \in J, \tag{2}$$

$$\sum_{m \in M_j} \sum_{t \in T_{jm}} \left(t + d_{jm}\right) x_{jmt} - \sum_{m \in M_l} \sum_{t \in T_{lm}} t x_{lmt} \le 0 \forall (j, l) \in B, \tag{3}$$

$$\sum_{j \in J} \sum_{m \in M_j} \sum_{t'=t-d_{jm}+1}^{t} v_{jmr} x_{jmt'} \le q_r \, \forall r \in R, \forall t \in T, \tag{4}$$

$$\sum_{j \in J} \sum_{m \in M_j} \sum_{t \in T_{jm}} u_{jmk} x_{jmt} \le o_k \quad \forall k \in K, \tag{5}$$

$$\sum_{m \in M_j} \sum_{t \in T_{jm}} (t + d_{jm}) x_{jmt} \le y_p \forall p \in P, j \in J_p, \tag{6}$$

$$z \ge y_p \forall p \in P \tag{7}$$

$$x_{jmt} \in \{0, 1\} \forall j \in J, m \in M_j, t \in T_{jm}.$$

In this formulation, constraints (2) ensure that every job is allocated in exactly one starting time and mode. Constraints (3) force precedence relations to be satisfied. Constraints (4) and (5) control the usage of renewable and nonrenewable resources, respectively. Finally, constraints (6) and (7) compute the completion time of individual projects and the final completion time of all projects.

3 Outline of proposed method

The major obstacles to generating feasible solutions for the MRCMPSP are the nonrenewable resources. While the use of renewable resources only impacts the duration of the projects, the use of nonrenewable resources can easily produce infeasible solutions. In our method, the first step is the definition of the job execution modes beforehand so that they respect the limits of nonrenewable resources. Once a set of feasible modes is obtained, a heuristic IP decomposition uses this set to iteratively build an initial feasible solution. This initial solution is then refined by a hybrid local search. This local search consists of a forward–backward improvement method hybridized with an IP model to change job execution modes and a biased rebuild procedure. More details on these methods are presented in the next two sections.

Figure 1 shows an outline of the developed solver.

4 Initial solution

As previously outlined, the initial solution is generated in two steps. First, a feasible set of execution modes is obtained. Then a matheuristic iteratively creates a feasible solution using the set of modes as guidance. The following subsections detail these two phases.

4.1 Obtaining a feasible set of execution modes

Unlike the works of Asta et al. (2014) and Geiger (2013), we employed an exact algorithm to search for feasible combinations of modes considering nonrenewable resources. This approach is similar to that used in Coelho and Vanhoucke (2011), with the difference that that earlier work modeled the problem as a Boolean satisfiability problem (SAT) and solved it using the DPLL algorithm (Davis and Loveland 1962). One drawback of the SAT approach is that an exponential number of clauses is needed to model nonrenewable resource constraints, and the authors encountered some difficulties in handling these models explicitly.

The problem of selecting these initial modes is denoted here by *ModeSel*, which is solved by an IP model. With this model we want to define, for each job, a mode such that the total nonrenewable resource consumption limits are always respected. The model itself is as hard as the m-dimensional knapsack problem and basically consists of the original model without the renewable resource constraints.

To present the *ModeSel* formulation, the same nomenclature (input data) from Sect. 2 is considered, with the following additional input data:

C:	set of selected paths between the start and end jobs of each project;
J_c^C:	set of jobs on a path $c \in C$;

The following decision and auxiliary variables are considered:

x_{jm} binary variable that is equal to 1 when the job j is executed in mode m and 0 otherwise;
t_j integer variable that indicates the start time of job j;
y_p integer variable that indicates the end time of project p.

The objective function of the proposed model, presented by Eq. (8), minimizes three parcels. To each parcel was assigned a weight that indicates its priority. The weights are defined hierarchically such that $\omega_1 \gg \omega_2 \gg \omega_3$. The first parcel is responsible for minimizing the sum of the makespan of all projects $p \in P$. The second and third parcels are responsible for minimizing the job durations. The second parcel differs from the third in that it prioritizes jobs that appear more often in the set of paths C.
Minimize

$$\omega_1 \sum_{p\in P} y_p + \omega_2 \sum_{c\in C} \sum_{j\in J_c} \sum_{m\in M_j} d_{jm}x_{jm} + \omega_3 \sum_{j\in J} \sum_{m\in M_j} d_{jm}x_{jm} \tag{8}$$

subject to

$$\sum_{m\in M_j} x_{jm} = 1 \; \forall j \in J, \tag{9}$$

$$\sum_{j\in J} \sum_{m\in M_j} u_{kjm}x_{jm} \le o_k \; \forall k \in K, \tag{10}$$

$$t_j + \sum_{m\in M_j} d_{jm}x_{jm} \le t_l \; \forall (j,l) \in B, \tag{11}$$

$$y_p \ge t_j \; \forall p \in P, \; \forall j \in J_p. \tag{12}$$

$$t_j \in \mathbb{Z}^+ \forall j \in J$$

$$x_{jm} \in \{0,1\} \forall j \in J, m \in M_j$$

Constraints (9) guarantee that only one mode is selected for each job, and constraints (10) ensure that the nonrenewable resource limits are satisfied. The start time of a job is defined through constraints (11) given that a job can only start after all its predecessors finishes. Finally, constraints (12) set the variables y_p to represent the end of each project $p \in P$ since for the last job (*dummy* job) the start time is equal to the end time (duration is zero).

4.2 Constructive algorithm

As stated earlier, the use of nonrenewable resources can easily produce infeasible solutions. To deal with this issue, we use the feasible set of execution modes provided by the *ModeSel* solution. This set guides the constructive schedule generation algorithm toward a feasible solution. Since the use of renewable resources only impacts the duration of projects, and these durations are not bounded, the constructive procedure always results in a feasible solution.

The constructive algorithm uses an IP heuristic based on decomposition to build an initial feasible solution, as follows. At each iteration, a time window is defined starting at time zero and an IP model is created and solved with the objective of allocating jobs within this window. The time windows are sequential and nonoverlapping. The algorithm finishes when all jobs are allocated. Original constraints, such as precedence and resource consumption, are considered in this model. Beyond the input data presented in Sect. 2, the following parameters are required by the model:

$\tilde{m}_j$: execution mode established by *ModeSel* for job j;
δ_{jm}: difference between (i) the sum of the first time slot after the window size, the duration of the job considering the mode allocation from *ModeSel*, and (ii) the current time instant, plus the duration of the job in the current mode;
φ_j: greatest distance from job j to dummy job that represents the end of the project of j, considering the execution mode established by *ModeSel*;
ϕ_p: ratio of maximum distance of all projects to maximum distance of each project p.

The following decision variables are defined:

x_{jmt}: binary variable that is equal to 1 when job j is processed in mode m and starts at time t, and 0 otherwise;
y_p: integer variable that indicates an estimation of the end time of project p – in general, the greater the number of allocated jobs of project p, the lower the value of y_p;
z: integer variable that indicates the maximum estimated end time between projects.

The objective function, given by Eq. (13), consists of three parcels. To each parcel was assigned a weight to indicate its priority. These weights are also defined hierarchically such that $\omega_1 \gg \omega_2 \gg \omega_3$. The first parcel is responsible for minimizing the estimate on the time of completion of each project. This estimate considers the jobs that cannot be allocated in the current time window. The second parcel is responsible for maximizing the number of allocated jobs, taking into consideration the priority established by δ for each one. Finally, the last parcel aims to minimize the overall completion time.
Minimize

$$\omega_1 \sum_{p\in P} y_p - \omega_2 \sum_{j\in J} \sum_{m\in M_j} \sum_{t\in T_{jm}} \delta_{jm}x_{jmt} + \omega_3 z \tag{13}$$

subject to

$$\sum_{m\in M_j}\sum_{t\in T_{jm}} x_{jmt} \leq 1 \forall j \in J, \tag{14}$$

$$\sum_{j\in J}\sum_{m\in M_j}\sum_{t'=t-d_{jm}+1}^{t} v_{rjm}x_{jmt'} \leq q_r \forall r \in R, \forall t \in T, \tag{15}$$

$$\sum_{j\in J}\left[\begin{array}{l}\displaystyle\sum_{m\in M_j}\sum_{t\in T_{jm}} u_{kjm}x_{jmt} + \\ u_{kj\tilde{m}_j}\left(1-\displaystyle\sum_{m\in M_j}\sum_{t\in T_{jm}} x_{jmt}\right)\end{array}\right] \leq o_k \forall k \in K, \tag{16}$$

$$\begin{array}{l}\displaystyle\sum_{m\in M_j}\sum_{t\in T_{jm}}(t+d_{jm})\,x_{jmt} - \sum_{m\in M_l}\sum_{t\in T_{lm}} t x_{lmt} \\ \displaystyle -\left(1-\sum_{m\in M_l}\sum_{t\in T_{lm}} x_{lmt}\right)(t+d_{lm})\end{array} \leq 0 \forall (j,l) \in B, \tag{17}$$

$$\sum_{m\in M_j}\sum_{t\in T_{jm}} x_{jmt} - \sum_{m\in M_l}\sum_{t\in T_{lm}} x_{lmt} \geq 0 \forall (j,l) \in B, \tag{18}$$

$$\left(1-\sum_{m\in M_j}\sum_{t\in T_{jm}} x_{jmt}\right)\left(\phi_p\varphi_j + d_{j\tilde{m}_j}\right) \leq y_p \forall p \in P, \forall j \in J, \tag{19}$$

$$z \geq y_p \forall p \in P \tag{20}$$

$$x_{jmt} \in \{0,1\} \forall j \in J, m \in M_j, t \in T_{jm}.$$

Constraints (14) ensure that each job is allocated at most once. Constraints (16) and (15) respectively ensure that the available amounts of nonrenewable and renewable resources are not extrapolated. Constraints (17) and (18) ensure the precedence relations between jobs. Two constraints are needed for this because it is possible for a job to remain unallocated in a solution. Thus, the precedence constraints should hold only for allocated jobs. Constraints (19) guarantee that if a job $j \in J_p$ is not allocated, then the end of project p will be based on an estimation. This estimation is calculated using the parameters ϕ_p and φ_j.

The pseudocode for generating the initial solution is presented in Algorithm 1. The algorithm takes as input the following data: (i) the set J of all jobs of the problem, (ii) the set of feasible execution modes, and (iii) the window size.

At each iteration the algorithm selects a subset of jobs that have the earliest start time in the range of the time window (line 4). After these jobs have been selected, a minimum amount of nonrenewable resources is reserved for the jobs that are not part of the subproblem (line 5). The *ModeSel* solution is used in this step. After that, the subproblem is created and solved (lines 6 and 7) and the solution of the subproblem is included in the main solution (line 8). The time window is then advanced (line 9). This process repeats until a feasible solution is generated and then returned (line 10).

It is important to note that a time limit is imposed on the solver. Thus, it is necessary to guarantee that a feasible solution will be generated. For this, a greedy feasible solution for each subproblem is created and passed to the IP solver

Algorithm 1: Constructive matheuristic

```
   Input: $J$, $\tilde{M}$, $wsize$
1  $S \leftarrow$ empty solution
2  $W \leftarrow (0, wsize)$
3  while (solution $S$ is not complete) do
4      $J' \leftarrow$ subset of jobs in $J$ not allocated to $S$ and eligible for
       window $W$
5      $R' \leftarrow$ nonrenewable resources not available for jobs $J'$,
       considering $\tilde{M}$
6      $P' \leftarrow$ model generated by $W$, $J'$, and $R'$
7      $S' \leftarrow$ (sub)optimal solution of $P'$
8      $S \leftarrow S \cup S'$
9      $W \leftarrow$ time window following $W$ with size equal to $wsize$
10 return $S$
```

before optimization. The greedy algorithm considers a topological ordering of the jobs and allocates them according to their renewable resource consumption. Jobs are allocated for processing in the modes defined by the *ModeSel* solution.

Another very important aspect of the algorithm concerns the time window. Defining the size of the time window itself is a tricky problem. The time window must be small enough to ensure that the generated IP models are easily solved and, at the same time, big enough to indicate relevant allocations.

5 Local search

We also employ an IP-based local search procedure that shares some similarities with the constructive algorithm. A time window is defined, and only jobs within this window are considered eligible to have their modes and starting times changed. This local search is combined with the forward–backward improvement (FBI) procedure, also called a double justification procedure (Valls et al. 2005). The FBI is a two-step improvement phase for a single project schedule generated by some other method. Finally, when the procedure is stuck in a local optimum, a biased rebuild method is executed to generate a new, perturbed, solution. These three methods are described in the following subsection.

5.1 Forward–backward improvement procedure

In the first step of the FBI (the forward pass), the jobs are right justified in the schedule, that is, except for the first and last jobs, all job allocations are shifted to the right, starting from the final job's immediate predecessors until the initial job's immediate successors. This step generates a right active schedule – a schedule where no activity can be finished without advancing some other activity or increasing the makespan (Valls et al. 2005). Since the final job is not shifted, the current makespan is held. If during the first step some slack is generated in the schedule, the second step tries to reduce

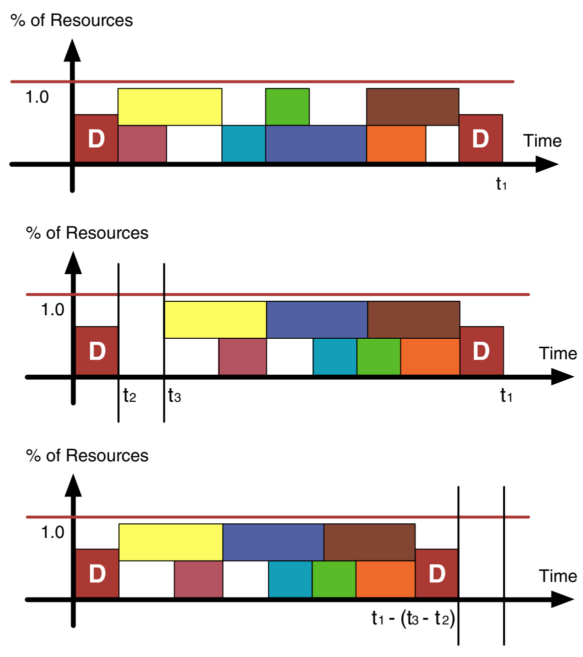

Fig. 2 Forward–backward improvement (FBI) example

the makespan by eliminating that slack. In this second step (the backward pass), jobs are analogously left justified in the schedule, that is, except for the initial job, all job allocations are shifted to the left, starting from the initial job's immediate successors until the final job. This step generates a left active schedule (a schedule where no activity can be started without violating constraints).

Figure 2 shows an example of this process. In this figure, the *dummy* jobs of a project (first and last jobs) are marked with a "D." The first graph represents the initial solution, and the next two graphs show the right active schedules.

In both steps the schedule is always feasible since the precedence relations and resource consumption constraints are satisfied. Each shift is performed using the serial generation scheme.

After the application of the FBI, some jobs might not have been shifted in any direction and could represent the problem's bottleneck. We propose a third improvement step that marks these stationary jobs and then randomly changes each marked job's mode and tries to perform a left shift using the serial generation scheme.

These three steps are performed while there is an improvement in the quality of the solution and until the time limit is reached.

5.2 Integer programming local search

The IP local search has as its objective to improve the solution with the optimization of subproblems, determined by a time window, sharing many similarities with the constructive algorithm. The only issue is that, by making changes in the middle or in the beginning of the schedule, one will not immediately improve the solution because the jobs that define the end of the projects will not be considered in the optimization. But this can be solved with the FBI procedure. Thus, the real objective of the IP local search is to consistently changes the solution for the FBI procedure to improve it.

To change the solution, the following calculated data are introduced in the model:

ω_{jmt}: priority of allocation of job j to mode m and time t.

The idea is to give the highest priority – lower value for ω_{jmt} – to the most varied allocations. Then current allocations have high values for ω_{jmt}. Other allocations multiply its priority by a random factor, inside [0, 1]. When the executed mode is also changed, this random factor is divided by 2. This approach generally led to very different solutions, always with the same objective value. In practice, this allowed the FBI procedure to keep improving the solution.

To generate this model, a lot of preprocessing is done to make it as small as possible. In this IP model, the following decision variable is defined:

x_{jmt}: binary variable that is equal to 1 when job j is processed in mode m and starts at time t, and 0 otherwise.

The objective function, given by Eq. (21), is responsible for changing the modes and start times of the jobs whenever possible. For that, each possible assignment is multiplied by its priority factor ω_{jmt}.

The constraints (22) are responsible for ensuring that each activity is allocated only once. The constraints (23) and (24) respectively ensure that the limits of nonrenewable and renewable resources are respected. Finally, constraints (25) ensure the precedence relations.

Minimize

$$\sum_{j\in J}\sum_{m\in M_j}\sum_{t\in T_{jm}} \omega_{jmt}x_{jmt} \tag{21}$$

subject to

$$\sum_{m\in M_j}\sum_{t\in T_{jm}} x_{jmt} \quad = 1 \forall j \in J, \tag{22}$$

$$\sum_{j\in J}\sum_{m\in M_j}\sum_{t\in T_{jm}} u_{kjm}x_{jmt} \quad \le o_k \, \forall k \in K, \tag{23}$$

$$\sum_{j\in J}\sum_{m\in M_j}\sum_{t'=t-d_{jm}+1}^{t} v_{rjm}x_{jmt'} \quad \le q_r \forall r \in R, \forall t \in T, \tag{24}$$

$$\sum_{m\in M_j}\sum_{t\in T_{jm}}(t+d_{jm})x_{jmt} - \sum_{m\in M_l}\sum_{t\in T_{lm}} t x_{lmt} \leq 0 \forall (j,l)\in B \quad (25)$$

$x_{jmt}\in\{0,1\}\forall j\in J, m\in M_j, t\in T_{jm}$.

5.3 Biased solution rebuild

To perturb the solution, a biased rebuild method is used. This method requires a guidance solution and works as follows. The jobs are put in a set sorted by start time in the guidance solution and a new solution, S', is created. Then the jobs are added to S' one at a time in the smallest start time that does not violate precedence or renewable resource constraints. Non-renewable resource constraints are always satisfied because the execution modes of the jobs in the guidance solution are used in the allocations.

The probability that each job will be chosen at a specific moment is given by a heuristic-biased stochastic sampling (HBSS) (Bresina and Bresina 1996). Any job can be chosen, but the first jobs have much greater probabilities of being selected. The chances of selecting a job is given by $f(r) = e^{-r}$, where r is the position of the job in the sorted set. After being chosen, the job will be added to S' only if it does not violate any precedence constraints. After being added to the solution, the job leaves the sorted set. The method returns after all jobs are added to S'.

5.4 Hybrid local search algorithm

The hybrid local search works on an iterated local search (Lourenco et al. 2003) fashion. In this context, the perturbation is implemented by the biased solution rebuild. The IP local search model may also be seen as a perturbation, except for the fact that it never worsens the solution.

The pseudocode of the local search is presented in Algorithm 2. The algorithm takes as input data (i) an initial solution; (ii) the maximum number of iterations; (iii) the maximum number of changes, i.e., the number of runs of the IP model per iteration; (iv) the minimum time window size; and (v) the maximum time window size.

Until the time limit is reached, the algorithm applies FBI while the solution is being improved (line 1). After that, the solution is changed using the IP model described in Sect. 5.2. The IPLS(S, W) function solves the IP model considering the solution S and the random time window W and then returns a modified solution (lines 5–7). These changes in the solution are made iteratively until the number of p iterations is reached. Subsequently, the FBI procedure is run again (line 8) to improve the solution. After some time of running without improvement, $iter$ may reach $iter_{max}$ and p is incremented (lines 14 and 15). Eventually, after many iterations without

Algorithm 2: Hybrid local search

```
   Input: $S, iter_{max}, change_{max}, wsize_{min}, wsize_{max}$
1  $S \leftarrow$ FBI($S$)
2  $S^* \leftarrow S$
3  $p \leftarrow iter \leftarrow 1$
4  while time limit is not reached do
5      for $i \leftarrow 0$ to $p$ do
6          $W \leftarrow$ random time window with random size in
           $[wsize_{min}, wsize_{max}]$
7          $S \leftarrow$ IPLS($S$, $W$)
8      $S \leftarrow$ FBI($S$)
9      if $S$ is better than $S^*$ then
10         $S^* \leftarrow S$
11         $p \leftarrow iter \leftarrow 1$
12     else
13         $iter \leftarrow iter + 1$
14     if $iter > iter_{max}$ then
15         $p \leftarrow change + 1$
16     if $p > change_{max}$ then
17         $p \leftarrow 1$
18         $S \leftarrow$ new biased solution using $S^*$
19 return $S^*$
```

improvement, the solution may be restarted using the biased rebuild method (lines 16–18). In this process, the best solution found so far is used to guide the generation of the new solution. The output of the method is a possibly improved feasible solution (line 19).

6 Computational experiments

All algorithms were coded in C++, and the IP models were solved using Gurobi 5.5. The code was compiled with GCC 4.6.2 using flag –O3, and all tests were run on an Intel Core i7 2.0 GHz computer with 16 GB RAM running Ubuntu Linux 12.04.

Table 1 shows the characteristics of the instances considered. The table displays the number of projects ($|P|$), the total number of jobs ($|J|$), the number of precedence relations between jobs ($|E|$), and the amount of global renewable ($|R_g|$), local renewable ($|R_l|$), and nonrenewable resources ($|K|$) of each instance. The table also presents the average job duration ($\langle d_{jm}\rangle$), the average number of execution modes per job ($\langle M_j\rangle$), and, finally, the average critical path duration of the projects ($\langle$CPD$\rangle$).

6.1 Formulation

The formulation presented in Sect. 2 was implemented in both Gurobi 5.5 and CPLEX 12.1. In Table 2 we present the results obtained with Gurobi, since Gurobi appeared to be slightly better in our tests. For these tests, the values for

Table 1 Characteristics of the instances

Instance	Dimensions						Average dimensions		
	$\|P\|$	$\|J\|$	$\|E\|$	$\|R_g\|$	$\|R_l\|$	$\|K\|$	$\langle d_{jm} \rangle$	$\langle \|M_j\| \rangle$	$\langle$CPD$\rangle$
A-1	2	24	36	1	2	4	5.19	2.67	14.50
A-2	2	44	80	1	2	4	4.90	2.82	22.50
A-3	2	64	116	1	2	4	5.48	2.88	33.50
A-4	5	60	90	1	5	10	5.01	2.67	14.20
A-5	5	110	200	1	5	10	5.62	2.82	23.00
A-6	5	160	290	1	5	10	5.07	2.88	25.60
A-7	10	120	180	2	0	20	5.33	2.67	16.80
A-8	10	220	400	2	0	20	5.26	2.82	24.60
A-9	10	320	580	1	10	20	5.32	2.88	29.60
A-10	10	320	580	1	10	20	5.48	2.88	30.70
B-1	10	120	180	1	10	20	5.23	2.67	12.90
B-2	10	220	400	2	0	20	5.46	2.82	23.90
B-3	10	320	580	1	10	20	5.38	2.88	29.50
B-4	15	180	270	1	15	30	5.35	2.67	15.80
B-5	15	330	600	1	15	30	5.33	2.82	22.53
B-6	15	480	870	1	15	30	5.39	2.88	31.13
B-7	20	240	360	1	20	40	5.15	2.67	15.35
B-8	20	440	800	2	0	40	5.27	2.82	23.65
B-9	20	640	1160	1	20	40	5.46	2.88	30.10
B-10	20	460	816	2	0	40	5.29	2.83	24.45
X-1	10	120	180	2	0	20	5.07	2.67	14.90
X-2	10	220	400	1	10	20	5.32	2.82	23.00
X-3	10	320	580	1	10	20	5.38	2.88	29.90
X-4	15	180	270	2	0	30	5.24	2.67	14.87
X-5	15	330	600	1	15	30	5.19	2.82	23.60
X-6	15	480	870	1	15	30	5.34	2.88	29.93
X-7	20	240	360	1	20	40	4.94	2.67	14.95
X-8	20	440	800	1	20	40	5.34	2.82	24.45
X-9	20	640	1160	1	20	40	5.24	2.88	28.90
X-10	20	450	798	1	20	40	5.30	2.82	24.10

$|T|$ were estimated as 110% of the TMS of the best known solution. The table displays the number of projects ($|P|$) and the total number of jobs ($|J|$) of each instance, as well as the dimensions of the model. The column *status* presents the status of the solver after the execution. In this column, *Feas.* stands for *feasible solution found*, *relax.* means that only the relaxation was solved, i.e., that an integer solution was not found, and finally, "–" means that not even the relaxation could be solved in the allowed runtime. The solvers ran out of memory for instances B-8, B-9, and B-10, and the model could not be generated. We omit the results for instances of set X because they are very similar in size to the instances of set B.

It is possible to verify from Table 2 that the IP solver was able to find the optimal solution within 1 s for the first three instances. For the others in which the model could be generated, either the solver found a poor solution or did not find a feasible solution at all, even after 7 h of runtime. For many instances, not even the linear relaxation was solved in the allowed runtime. This confirms the hardness of the problem, as even with a compact formulation it is very hard to deal with the problem using the current generation of commercial linear programming solvers.

6.2 Proposed method

The developed method was run in parallel in three threads, each one with different parameter values. Although the implemented algorithm is clearly sequential, the competition allowed the use of up to four threads. The parameter values were obtained after several empirical tests and are presented in Table 3. Basically, the size of the time windows is the most critical parameter. After several runs, we observed that subproblems with up to 40 timeslots could be solved by both CPLEX and Gurobi in an acceptable amount of time. Larger time windows, however, generated subproblems that required longer execution times to be solved. We also took some pre-

Table 2 Experiments with IP formulation

Instance	Dimensions		Model dimensions			Status	Gap	Time(s)
	$\|P\|$	$\|J\|$	Variables	Constrs	Nonzeros			
A-1	2	24	743	175	9734	Optimal	0.0%	0.01
A-2	2	44	2149	290	34,452	Optimal	0.0%	0.32
A-3	2	64	4425	373	75,010	Optimal	0.0%	0.08
A-4	5	60	12,114	778	138,597	Feas.	12.5%	25,200.02
A-5	5	110	52,239	1490	786,609	Feas.	69.7%	25,200.01
A-6	5	160	75,688	1696	1,020,878	Relax.	–	25,200.01
A-7	10	120	194,682	1642	2,707,008	Feas.	70.7%	25,200.04
A-8	10	220	197,988	1434	2,726,661	Feas.	79.1%	25,200.04
A-9	10	320	224,475	4264	3,106,213	Relax.	–	25,200.03

Table 2 continued

Instance	Dimensions		Model dimensions			Status	Gap	Time(s)
	$\|P\|$	$\|J\|$	Variables	Constrs	Nonzeros			
A-10	10	320	911,172	12,855	14,181,884	–	–	25,200.05
B-1	10	120	114,865	4786	1,590,869	Feas.	64.4%	25,200.03
B-2	10	220	279,542	1684	3,729,734	Feas.	79.3%	25,200.11
B-3	10	320	519,853	8125	8,500,140	Relax.	–	25,200.05
B-4	15	180	617,183	22,208	9,000,533	–	–	25,200.00
B-5	15	330	777,307	15,424	11,217,583	–	–	25,200.00
B-6	15	480	1,287,472	17,972	20,412,202	–	–	25,200.00
B-7	20	240	514,332	19,393	6,470,080	–	–	25,200.00
B-8	20	440	–	–	–	–	–	–
B-9	20	640	–	–	–	–	–	–
B-10	20	460	–	–	–	–	–	–

Table 3 Parameters used for tests

Thread	Initial solution	Local search			
	$wsize$	$iter_{max}$	$change_{max}$	$wsize_{min}$	$wsize_{max}$
1	40	10	15	10	40
2	40	20	10	10	40
3	20	40	20	10	20

cautions and added a thread with a smaller time window size (third thread) to ensure that the solver returned a solution in the expected time. The final algorithm is not very sensible to the other parameters because they only impact the local search phase.

Table 4 shows the results obtained after 50 runs with the proposed approach within 300 s of runtime. In the qualification phase of the competition, our method was ranked second among 16 teams. In the finals, it was ranked third. Figure 3 shows the gap between the results of the 50 runs for all instances in several boxplots. As can be seen, our approach performs superbly in some instances, where the generated solutions are almost 8% better than the results from the MISTA Challenge. We believe that larger subproblems tend to lead to better solutions. Unfortunately, solving larger problems is too slow for the current generation of IP solvers. As we keep researching new ways of addressing these sub-

Table 4 Best and average results after 50 runs of algorithm sided with best results from MISTA

Inst.	Best		Average		Std.Dev.		MISTA		≤ MISTA?
	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	
A-1	1	23	1	23	0	0	1	23	Yes
A-2	2	41	2	41	0	0	2	41	Yes
A-3	0	50	0	50	0	0	0	50	Yes
A-4	68	50	69	42	0	4	65	42	
A-5	154	104	157	103	2	1	153	105	
A-6	151	94	160	95	4	1	147	96	
A-7	626	194	633	195	6	0	596	196	
A-8	281	147	298	147	9	1	302	155	Yes
A-9	212	127	217	124	3	2	223	119	Yes
A-10	983	309	1017	309	10	2	969	314	
B-1	358	131	364	130	3	1	349	127	
B-2	431	159	450	159	4	1	434	160	Yes
B-3	585	196	609	197	7	3	545	210	
B-4	1435	294	1516	292	25	2	1274	289	
B-5	867	254	884	251	6	2	820	254	
B-6	970	224	1096	227	21	3	912	227	
B-7	876	234	889	239	10	3	792	228	

Table 4 continued

Inst.	Best		Average		Std. Dev.		MISTA		≤ MISTA?
	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	
B-8	3001	520	3128	518	31	5	3176	533	yes
B-9	4753	741	4825	744	49	3	4192	746	
B-10	3123	430	3175	432	25	2	3249	456	yes
X-1	392	142	401	143	4	1	392	142	yes
X-2	416	167	421	168	4	1	349	163	
X-3	332	177	350	181	10	2	324	192	
X-4	980	209	996	209	12	1	955	213	
X-5	1904	369	1944	371	16	2	1768	374	
X-6	821	237	849	240	12	2	719	232	
X-7	909	232	923	236	7	4	861	237	
X-8	1389	281	1425	282	20	2	1233	283	
X-9	3945	639	3998	650	39	4	3268	643	
X-10	1718	377	1755	378	20	4	1600	381	

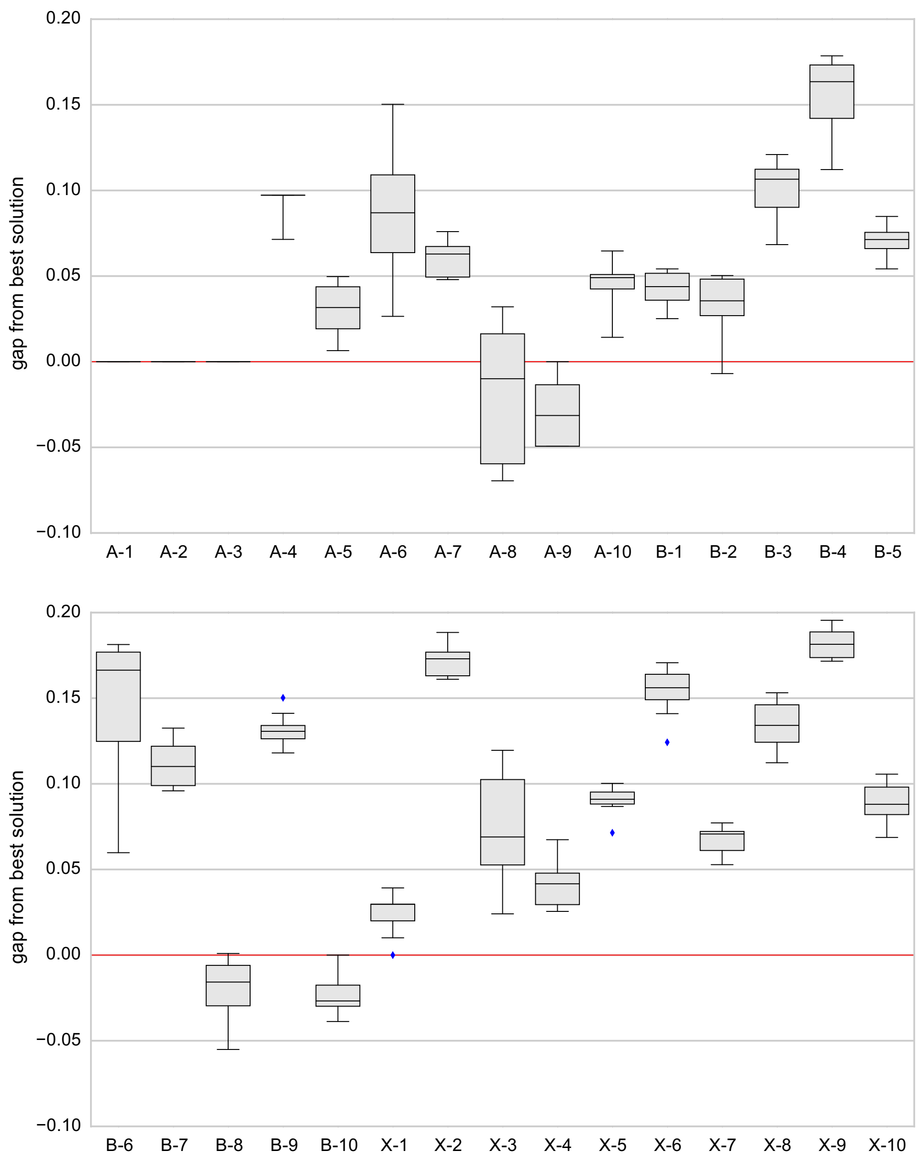

Fig. 3 Gap of the solution value of 50 runs from the MISTA Challenge best solutions

problems, we hope that, given the continued evolution of these solvers, in the near future it will be possible to solve larger subproblems. In this context, matheuristics such as the one presented in this paper are very desirable, as was pointed out by Fischetti et al. (2010).

7 Conclusions

This work presented a formulation and a hybrid algorithm with several IP-based components for the MRCMPSP:

- Mode-selection IP model;
- IP constructive algorithm;
- Forward–backward improvement (FBI) procedures;
- IP local search algorithm; and
- Biased rebuild solution algorithm.

The proposed approach was able to produce very good solutions and ranked third in the MISTA 2013 Challenge (Wauters et al. 2014). Among the winners, our approach was the only one to include IP techniques within the method and was still competitive considering the short runtime of 5 min. Taking into consideration the recent and continuous evolution of IP commercial solvers, approaches that include these techniques are desirable (Jünger et al. 2010).

Finally, we have some recommendations for future work, as the proposed approach still has room for improvement. For instance, the objective function relies on estimations that should be further analyzed. It is very likely that, by finding more suitable objective functions, the same algorithms will lead to better solutions. Another direct improvement would be to add the neighbor structures used by the other finalists of the competition, which include expected movements such as switch jobs, changing the order of jobs, and others.

Acknowledgments The authors thank CNPq and FAPEMIG for supporting this research.

References

Alba, E., & Chicano, J. F. (2007). Software Project Management with GAs. *Information Sciences*, *177*, 2380–2401.

Artigues, C., Demassey, S., & Néron, E. (2013). *Resource-constrained project scheduling: Models, algorithms, extensions and applications*. Hoboken: Wiley.

Asta, S., Karapetyan, D., Kheiri, A., Özcan, E., & Parkes, A. (2014). *Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem*. Technical report.: School of Computer Science, University of Nottingham.

Blazewicz, J., Lenstra, J., & Rinnooy Kan, A. (1983). Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, *5*(1), 11–24.

Bresina, J., Bresina, L. (1996). Heuristic-biased stochastic sampling. In *AAAI-96 Proceedings* (pp. 271–278).

Coelho, J., & Vanhoucke, M. (2011). Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *European Journal of Operational Research*, *213*(1), 73–82.

Davis, M., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*(7), 394–397.

De Marco, A. (2011). *Project management for facility construction*. Heidelberg: Springer.

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project scheduling: A research handbook*. Leuven: Kluwer Academic Publishers.

Fischetti, M., Lodi, A., & Salvagnin, D. (2010). just mip it!. In V. Maniezzo, T. Stützle, & S. Voü (Eds.), *Matheuristics, annals of information systems* (Vol. 10, pp. 39–70). New York: Springer.

Geiger, M. (2013). Iterated variable neighborhood search for the resource constrained multi-mode multi-project scheduling problem. some comments on our contribution to the MISTA 2013 challenge. *Multidisciplinary International Scheduling Conference (MISTA) 2013 Proceedings 27–29* (pp. 807–811).

Hansen, P., & Mladenović, N. (1997). Variable neighborhood search. *Computers and Operations Research*, *24*(11), 1097–1100.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, *49*(5), 433–448.

Józefowska, J., & Weglarz, J. (2006). *Perspectives in modern project scheduling*. New York: Springer.

Jünger, M., Liebling, T., Naddef, D., Nemhauser, G., Pulleyblank, W., Reinelt, G., et al. (2010). *50 years of integer programming 1958–2008*. Heidelberg: Springer.

Kelley, Jr J.E., Walker, M.R. (1959). Critical-path planning and scheduling. In: *Papers Presented at the December 1–3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference* (pp. 160–173). ACM, New York, NY, USA, IRE-AIEE-ACM '59 (Eastern).

Kerzner, H. (2009). *Project management: A systems approach to planning, scheduling, and controlling* (10th ed.). New York: Wiley.

Klein, R. (2000). *Scheduling of resource-constrained projects*., Operations research/computer science interfaces series Norwell: Kluwer Academic.

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In J. Wglarz (Ed.), *Project scheduling, international series in operations research & management science* (Vol. 14, pp. 147–178). New York: Springer.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, *174*(1), 23–37.

Kolisch, R., & Sprecher, A. (1997). PSPLIB—a project scheduling problem library. *European Journal of Operational Research*, *96*(1), 205–216.

Kone, O., Artigues, C., Lopez, P., & Mongeau, M. (2011). Event-based MILP models for resource-constrained project scheduling problems. *Computers and Operations Research*, *38*, 3–13.

Lourenco, H. R., Martin, O. C., & Stutzle, T. (2003). *Iterated Local Search*. Boston: Kluwer Academic Publishers.

Maniezzo, V., Stutzle, T., & Vo, S. (2010). *Matheuristics: Hybridizing metaheuristics and mathematical programming* (Vol. 10). Leuven Belgium: Springer.

Möhring, R. H., Schulz, A. S., Stork, F., & Uetz, M. (2003). Solving project scheduling problems by minimum cut computations. *Management Science*, *49*(3), 330–350.

Pritsker, A. A. B., Watters, L. J., & Wolfe, P. M. (1969). Multi project scheduling with limited resources: A zero-one programming approach. *Management Science*, *3416*, 93–108.

Valls, V., Ballestín, F., & Quintanilla, S. (2005). Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, *165*(2), 375–386.

Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Vanden Berghe, G., & Verstichel, J. (2014). The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, *2014*, 1–13.

Weglarz, J. (1999). *Project scheduling: Recent models, algorithms, and applications*., International series in operations research & management science Boston: Kluwer Academic Publishers.