Techniques to solve a Multi-Mode Resource Constrained Project Scheduling Problem with energy saving

André Renato Villela da Silva

Institute of Science and Technology
Federal Fluminense University
arvsilva@id.uff.br

Abstract. This paper deals with a new variant of the classical Resource-Constrained Project Scheduling Problem (RCPSP). In this variant, called by Multi-Mode Resource-Constrained Project Scheduling Problem with energy (MRCPSP-energy), each job has different execution modes where duration and energy consumption are conflicting. To make a job take less time to finish, it is necessary to spend more energy in its execution and vice-versa. This situation happens in several mechanized operations where the machines can operate in distinct energy saving manners.
In order to tackle this problem, a Mixed Integer Programming (MIP) is proposed as well as some tightening constraints. For large instances, a metaheuristic based on Ant Colony Optimization technique was tested. The obtained results show that the formulation provided significant bounds. The heuristic results are also competitive with other results.

Keywords: MRCPSP, energy consumption, MIP formulation, Ant Colony Optimization, metaheuristics

1 Introduction

The Project Scheduling Problem (PSP) is a class of combinatorial optimization problems where a project is composed of n jobs (activities) that must be executed. Each job i has a duration $d_i \geq 0$ and may have precedence over job j, i.e., i must be completely executed before job j starts. One of the most widely researched variants is the Resource-Constrained Project Scheduling Problem (RCPSP). Suppose a set $K = \{1, .., R\}$ of different types of renewable resources, a job i needs $q_{ik} \geq 0$ units of resource k $\forall (k \in K)$ per time unit during its execution. After its finish, all allocated resources are freed and can be used by other jobs, since these resources are renewable. The objective is to find the optimal start times of every job so the project makespan is minimum. It is important to notice that the jobs are non-preemptive and the total amount of all available resources is known, a priori.

Usually, the jobs and their precedence are represented by a Directed Acyclic Graph (DAG), like in Figure 1. The numbers above the vertices indicate the duration and the resource quantity needed by the job, respectively. By the sake of simplicity, only one type of resource is shown. The solution makespan is equal to 13 time units. Two artificial jobs are included representing the project start and the project finish, when necessary. These jobs, called job 0 and job $n+1$, have duration equal to 0 and make the mathematical modeling easier.

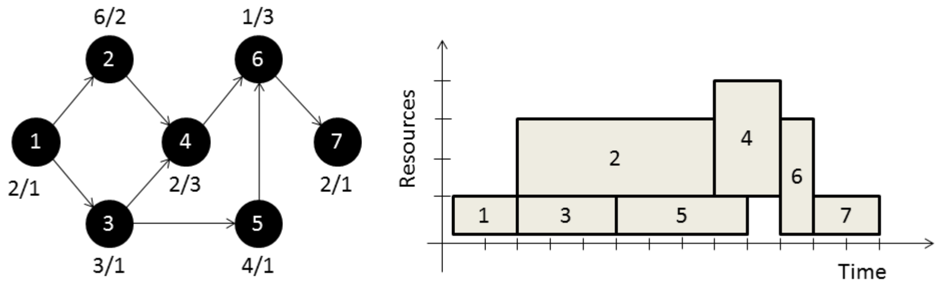

Fig. 1. A DAG representing a project with 7 jobs and only one resource type. The solution has makespan equal to 13 time units. The numbers above the jobs are their duration and the resource units needed.

Another frequently researched variant is the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP). In this variant, the jobs may have different execution modes. For each mode m, a job i has a duration d_i^m and a resource requirement of q_{ik}^m units of resource k. The objective is still the same (minimize the project makespan), but now the solution must also specify which execution mode was chosen for each job.

In [18], a variant concerning energy saving is presented. This variant was denoted by MRCPSP-energy and addresses the situation where the jobs require energy to start. The smaller the amount of energy applied to the job the longer it takes to finish. So, each job has multiple execution modes under specific duration and energy need. The objective is to minimize the project makespan and the total energy spent simultaneously, which is conflicting.

MRCPSP-energy deals mainly with situations where jobs are machine performed. Modern machines often provide some operation profiles, including energy-saving modes. Laptops are perfect examples of that. The user can define a power saving mode that limits the CPU speed, for example. This way, the battery will last longer but programs will run slower. Nowadays, energy is a very important and, sometimes, costly resource. In that way, many combinatorial optimizations problems deal with minimization of energy consumption. Obviously, MRCPSP-energy, as a PSP variant, still concerns about finishing all jobs as soon as possible. The metric used by MRCPSP-energy is called “efficiency”, which can be expressed by Eq. 1 to a project p. $lbCPM(p)$ represents the minimum project makespan, using a Critical Path Method (CPM) and considering the shortest duration for each job. $e_{min}(p)$ is the sum of necessary energy by the minimum energy mode of every job in p. $makespan(p)$ and $e_{total}(p)$ are, respectively, the actual makespan and total energy of project p. As the product $makespan(p) * e_{total}(p)$ gets lower, the efficiency increases.

$$efficiency(p) = (lbCPM(p) * e_{min}(p))/(makespan(p) * e_{total}(p)) \qquad (1)$$

The remainder of this work is organized as follows. Section 2 presents a brief literature overview about the RCPSP and variants. In Section 3, a Mixed-Integer Programming (MIP) formulation and a metaheuristic are proposed for the MRCPSP-energy. Some computational experiments are described in Section 4. Finally, in Section 5 there are some concluding remarks.

2 Literature overview

RCPSP and variants have been studied for decades. There are some surveys that seek to classify or give a general notation for problem models as presented in [1, 4, 10]. Since the RCPSP and MRCPSP are NP-hard [1], no efficient algorithm is known for medium and large instances. The main approaches include constructive heuristics [2, 11], metaheuristics [3, 5, 8, 9, 17], hybrid techniques [6, 20] and exact methods [7, 14, 17, 21].

In [18], the MRCPSP-energy was proposed as well as a benchmark set of instances, including 2040 projects with 30, 60, 90 or 120 jobs. The instances are modified versions of the well-known PSP-library, adapted to MRCPSP-energy input format. For these instances, a lower bound for makespan and for the total energy spent is known. Thus, an initial lower bound for MRCPSP-energy can be quickly computed. This bound, however, is very poor since many jobs cannot be executed with the minimum duration and energy cost simultaneously. There are no published algorithms to solve MRCPSP-energy, according to [16].

3 Proposed methods

The Mixed Integer Programming (MIP) is a combinatorial optimization technique where the problem is modeled by integer and binary variables which represent a solution. Some sets of constraints are imposed to these variables creating a feasible region inside the solution space. Usually, this formulation is executed by a kind of software, called solver, which implements several very efficient solving methods.

Table 1 presents the proposed variables and constants. In the formulation, M represents the available execution modes, T represents the planning horizon in whose time units the jobs must be executed and R represents the set of resources.

Table 1. MIP variables and constants

Symbol	Type	Description
$y_{i,t}^m$	Bin. Variable	Equal to 1 if job i starts at time t with mode m; 0, otherwise
a_i^m	Bin. Variable	Equal to 1 if job i is executed with mode m; 0, otherwise
e_{total}	Int. Variable	Total energy consumed
d_i	Int. Variable	Duration of job i
s_i	Int. Variable	Starting time of job i
f_i	Int. Variable	Finishing of job i
E_i^m	Constant	Energy needed to execute job i in mode m
D_i^m	Constant	Duration of job i in mode m
C_i^r	Constant	Amount of resources r to execute job i per time unit
Q^r	Constant	Total of available resources r

$$min \qquad e_{total} * f_{n+1} \tag{2}$$

$$s.t. \qquad \sum_{m=1}^{M}\sum_{t=0}^{T} y_{i,t}^m = 1 \qquad \forall_{i=0,..,n+1} \tag{3}$$

$$a_i^m = \sum_{t=0}^{T} y_{i,t}^m \quad \forall_{m=1,..,M} \quad \forall_{i=0,..,n+1} \tag{4}$$

$$d_i = \sum_{m=1}^{M} D_i^m * a_i^m \qquad \forall_{i=0,..,n+1} \tag{5}$$

$$s_i = \sum_{m=1}^{M}\sum_{t=0}^{T} t * y_{i,t}^m \qquad \forall_{i=0,..,n+1} \tag{6}$$

$$f_i = s_i + d_i \qquad \forall_{i=0,..,n+1} \tag{7}$$

$$e_{total} = \sum_{m=1}^{M}\sum_{i=0}^{n+1} E_i^m * a_i^m \tag{8}$$

$$s_j \geq f_i \qquad \forall_{(i,j)\in E} \tag{9}$$

$$\sum_{i=0}^{n+1}\sum_{m=1}^{M}\sum_{t'=t-D_i^m+1}^{t} C_i^r * y_{i,t}^m \leq Q^r \quad \forall_{r=1,..,R} \quad \forall_{t=0,..,T} \tag{10}$$

Eq. (2) defines the objective, which is to minimize the project makespan (f_{n+1}) and the total energy consumed. The available solvers usually do not handle well non-linear constraints or objective-functions as (1). Their performance (in terms of computational time spent) tends to be worse with non-linearized models, although some of them can actually solve such formulations. The proposed MIP replaces (1) by Eq. (2), which is mathematically equivalent in the optimization process. Every job must be executed once, as stated by constraints (3). The chosen mode for each job is given by constraints (4). The duration, starting and finishing time of each job are computed by constraints (5), (6), (7), respectively. The total amount of energy is modeled by constraint (8). Constraints (9) ensure the precedence between jobs i and j. The resources constraints are defined by (10).

Even using Eq. (2), the proposed objective-function continues to be non-linear. As stated before, most of the commercial solvers do not receive well such constraints, since multiplication of two or more integer variables is indeed very difficult to optimize. In order to overcome this issue, some modifications are suggested. First of all, f_{n+1} is equal to s_{n+1} because the artificial job $n+1$ was created only to be the last scheduled job and has duration always equal to 0. Next, s_{n+1} can be replaced by $\sum_{m=1}^{M}\sum_{t=0}^{T} t*y_{i,t}^{m}$ according to constraints (6). All modes of job $n+1$ are equivalent, so it is possible to use anyone, e.g., mode 1. The objective-function is now $\sum_{t=0}^{T} t*y_{n+1,t}^{1}*e_{total}$. Each parcel of this summation is a product between a binary and an integer variable, respectively, $y_{n+1,t}^{1}$ and e_{total}, as well as a coefficient t. Suppose $z_t = y_{n+1,t}^{1} * e_{total} \quad \forall_{t=0,..,T}$. The objective can be rewritten as $min \sum_{t=0}^{T} t * z_t$. Following, the linearized formulation is presented, where $\bar{E}$ is an upper bound for e_{total}, i. e., the maximum energy that could be used by a feasible solution.

$$min \qquad \sum_{t=0}^{T} t * z_t \tag{11}$$

$$(3) - (10)$$

$$z_t \geq 0 \quad \forall_{t=0,..,T} \tag{12}$$

$$z_t \leq \bar{E} * y_{n+1,t}^{1} \quad \forall_{t=0,..,T} \tag{13}$$

$$z_t \geq \bar{E} * (y_{n+1,t}^{1} - 1) + e_{total} \quad \forall_{t=0,..,T} \tag{14}$$

Constraints (13) bound the z_t to be at most $\bar{E}$. If $y_{n+1,t}^{1} = 1$, then $\bar{E} \geq z_t \geq e_{total}$. As MRCPSP-energy is a minimization problem, z_t should be as low as possible. On the other hand, if $y_{n+1,t}^{1} = 0$, then $z_t = 0$. By constraints (3), there is only one t^{+}, representing the starting time of job $n+1$, such that variable $z_{t^+} > 0 (z_{t^+} = e_{total})$. This way, the solution value if given by $t^{+} * e_{total}$. The lower the value of t^{+}, the better the solution is.

The formulation still has a poor linear relaxation. In order to improve this lower bound, a new set of integer variables $x_t \quad \forall_{t=0,..,T}$ is included. These variables are equal to 1 if the project are ongoing at time unit t and are equal to 0, if the project has already been finished at time t. The following constraints are not necessary to make the linearized formulation correct. However, they make it tighter.

$$x_t \geq x_{t+1} \quad \forall_{t=0,..,T-1} \tag{15}$$

$$x_t \leq \sum_{i=0}^{n+1} \sum_{m=1}^{M} \sum_{t'=t+1}^{T} y_{i,t'}^m \quad \forall_{t=0,..,T-1} \tag{16}$$

$$z_t \geq \sum_{i=0}^{n+1} \sum_{m=1}^{M} E_i^m * y_{i,t}^m \quad \forall_{t=0,..,T} \tag{17}$$

$$f_{n+1} = \sum_{t=0}^{T} x_t \tag{18}$$

$$z_t \leq \bar{E} * x_t \quad \forall_{t=0,..,T} \tag{19}$$

$$z_t \geq \bar{E} * (x_t - 1) + e_{total} \quad \forall_{t=0,..,T} \tag{20}$$

It is important to notice that, if constraints (19) and (20) replace constraints (13) and (14), several variables z_t should be equal to e_{total}, since $x_t = 1$ until the project is finished. Thus, the objective-function (11) can omit the coefficients t, which implies in the final formulation version as follows.

$$\begin{aligned} &min \qquad && \sum_{t=0}^{T} z_t && (21) \\ &s.t. && (3)-(10),(12),(15)-(20) \end{aligned}$$

Exacts methods, like the proposed MIP formulation, may take a very long computational time even for medium-size instances. In order to produce good solutions in acceptable time, an Ant Colony Optimization algorithm is proposed (ACO) [12, 15, 19]. ACO is a bio-inspired metaheuristic, where virtual ants represents the solution generation process. In nature, some ants of a colony are responsible for collecting food and bringing it back to the colony. Along the path that is chosen by the ant, some pheromone is expelled by the ant and marks the route traveled so that other ants can follow this path.

The paths that are more traveled are more likely to be used than those paths by which few ants have passed. Because the pheromone dissipates as time passes, shorter (better) paths retain higher amounts of pheromone. Thus, the ants adaptively perform intensification around a local optimal solution tending to make it even better. Algorithm 1 presents a very simple pseudo-code of the proposed ACO. After the initial pheromone is defined (line 1), the ACO keeps creating ants and updating the pheromone (lines 4-10), until the stopping criterion is met.

The most important mechanism of ACO is the pheromone management. The MRCPSP-energy is not a routing problem, so associating a pheromone value to each edge does not making any sense. Instead, the pheromone is associated to each job and execution mode. Hence, if a job i and mode m is chosen by some

Algorithm 1 ACO(input: instance data)

```
 1: $InitiatePheromone()$
 2: $bestAnt \leftarrow NULL$
 3: while stop criterion is not met do
 4:     $A \leftarrow ReleaseAnts(nAnts)$
 5:     for all $a_i \in A$ do
 6:         if $value(a_i) < value(bestAnt)$ then
 7:             $bestAnt \leftarrow a_i$
 8:         end if
 9:     end for
10:     $UpdatePheromone(A, bestAnt)$
11:     $DestroyAnts(A)$
12: end while
13: return $bestAnt$
```

Table 2. ACO internal parameters

Name	Value	Description
$nAnts$	300	Number of released ants at each ACO iteration
p_0	20.0	The initial pheromone level on all jobs and execution modes
$maxPheromone$	8.0	Maximum pheromone level released by a ant(best ant)
$dissipRate$	0.1	Reduction (10%) of pheromone levels at each ACO iteration

ant, the pheromone level p_i^m is increased by the pheromone released by some ant. All initial levels have the same value p_0.

The $UpdatePheromone(A, bestAnt)$ procedure (line 10) takes the entire population of ants and computes the level of pheromone ph_j released by each ant j (line 4). This level is proportional to the ant's solution value. In other words, the worst ant of a population releases no pheromone and the best one releases $maxPheromone$ (internal parameter). In order to respect the job scheduling in ant j, ph_j is gradually decreased until the last job is scheduled. The proposed decreasing rate is ph_j/n, where n is the number of jobs. After processing all ants, the ACO computes the pheromone dissipation. Each pheromone level p_i^m is reduced by $dissipRate$. Table 2 summarizes the internal parameters and their proposed values. These values were defined after preliminary test with several distinct configurations.

4 Computational experiments

Some computational experiments were executed in order to measure the performance of both mathematical formulation and metaheuristic proposed. The PSPLIB-ENERGY [16] is a benchmark composed of projects with 30, 60, 90 and 120 jobs. There are 480 instances with 30, 60 or 90 jobs and 600 instances with 120 jobs. These instances are divided in groups of 10 instances each. All groups have instances with the same size.

Table 3. Main results produced by the MIP formulation. Only projects with 30 jobs.

Groups	Opt	Avg. Opt(s)	Avg. Gap (%)	Groups	Opt	Avg. Opt(s)	Avg. Gap (%)
1-4	40	23.0	-	25-28	23	279.8	17.5
5-8	26	166.9	8.0	29-32	17	134.3	23.7
9-12	20	360.7	14.0	33-36	40	19.4	-
13-16	16	180.4	16.8	37-40	30	76.8	8.8
17-20	40	41.6	-	41-44	18	151.0	16.5
21-24	28	131.1	6.5	45-48	11	41.2	14.4

The MIP formulation was executed with CPLEX 12.7.1. Although CPLEX is commercial software, researchers and academics can use it for free and unrestrictedly The proposed ACO was implemented in language C++ and compiled with g++ 4.7.1. The computer configuration includes a processor Intel I7-3630QM, 8GB RAM and operating system Windows 8.0.

In the first experiment, the CPLEX solver executed the formulation using all 480 instances with 30 jobs each. Exact approaches like MIP are highly time-consuming methods. Even for small instances, they can take a very large computational time. The time limit given to the optimization process was equal to 1800 seconds. Table 3 presents some results in a condensed form, by groups of instances. Column “Groups” indicates the respective groups. “Opt” is the number of optimal solutions found. Columns “Avg. Opt(s)” and “Avg. Gap(%)” indicate, respectively, the total time spent in instances where the optimal solution is known and the average gap where the optimal is unknown. This gap is the difference between the incumbent solution and the best lower (dual) bound so far.

The formulation had a significant performance finding 309 optimal solutions (almost 65% of the tested instances). The average final gap is significant for some groups (groups 29-32 and 25-28, for example), indicating that these instances are harder, from the practical point of view. Among the instances in which the solver was able to find the optimal solution, the computational time spent was relatively low - approximately 2 minutes per instance. The formulation was tested in instances with 60 jobs and time limit of 3600 seconds, but the gaps were much larger and, in some cases, not even a single feasible solution was found.

Suppose $eff_{base} = lbCPM * e_{min}$. Values of eff_{base} are a very unreal and unattainable bound because there is an explicit trade-off between job duration and energy consumption. Indeed, to find good dual bounds for MRCPSP-energy is difficult. CPLEX can help with this, by executing some preprocessing algorithms. When exploring the Branch-and-Bound root node, also called “node 0”, CPLEX computes the problem linear relaxation and generates cutting planes in order to incorporate then to the given formulation. Such cutting planes normally strengthen the formulation but might take a considerable time to be produced. During this initial phase, some modifications on original constraints can also be performed and sometimes a feasible solution might be found. Table 4 shows some projections obtained by CPLEX preprocessing on node 0 for a subset of instances. Columns "DB%" indicate the maximum efficiency possible after node 0. The gap between CPLEX bounds at node 0 are provided in columns "gap%") and the average preprocessing time (in seconds) are shown in columns "T(s)". CPLEX received a time limit of 1800 seconds.

Table 4. Maximum efficiency possible after processing node 0 (column "DB%"). Gap between CPLEX bounds after node 0 (column "gap%"). Average preprocessing time (seconds).

Group	30 jobs			60 jobs			90 jobs			120 jobs		
	DB%	gap%	T(s)	DB%	gap%	T(s)	DB%	gap%	T(s)	DB%	gap%	T(s)
1	73.3	12.3	4.1	78.2	30.1	37.0	77.4	53.0	175.9	75.0	79.0	1040.8
2	75.2	4.7	3.2	81.0	4.4	11.3	83.9	6.4	27.5	80.2	63.8	555.6
3	77.8	1.6	1.7	80.8	3.6	9.2	83.9	1.5	13.7	85.6	35.0	191.3
4	77.1	0.0	1.4	84.1	0.4	3.3	84.5	0.4	6.2	84.6	25.9	163.8
5	67.7	57.2	22.6	72.1	92.4	197.5	76.6	101.4	353.8	85.5	15.2	76.3
6	73.5	19.6	9.0	78.6	30.7	26.7	82.5	33.6	86.2	82.8	-	1800.0
7	74.3	6.7	3.9	81.3	4.2	13.2	84.0	5.3	25.1	77.3	-	984.9
8	77.9	0.3	1.3	80.4	0.6	5.7	85.1	0.3	8.2	79.6	87.3	579.0
9	62.9	76.3	63.4	66.5	102.7	467.3	63.8	-	1161.6	82.4	56.4	233.5
10	75.0	26.5	14.2	80.4	35.8	43.9	84.1	34.1	131.9	84.7	40.8	206.8
11	75.0	11.1	8.4	81.1	10.2	19.6	82.7	14.8	52.6	77.2	-	1800.0
12	77.5	1.0	3.1	81.5	0.2	5.7	83.8	0.0	9.4	65.7	-	1628.0
13	56.8	74.8	78.3	52.4	87.8	961.5	56.4	-	1800.0	73.6	110.4	989.1
14	72.6	33.8	14.2	76.4	39.7	56.3	82.7	36.9	181.3	78.8	71.9	675.6
15	79.1	13.8	8.5	83.1	13.6	22.3	83.6	15.8	54.9	84.5	43.3	256.3
16	78.1	0.3	2.0	79.5	0.5	6.7	81.0	1.6	18.0	97.5	-	1800.0
17	70.8	13.5	6.3	76.7	29.1	43.0	79.1	43.8	188.2	55.2	-	1800.0
18	73.0	6.7	4.0	81.1	6.4	16.0	82.7	7.8	26.1	64.9	77.8	1611.8

The computational time required by the preprocessing phase was relatively small, except for some groups of 90 and 120 jobs, which exceeded 1000 seconds. In 14 instances, the preprocessing was aborted due to time limit without producing a valid bound. In some cases, the gap was so small that the optimization could finish exactly at node 0. The CPLEX preprocessing found 45, 38, 37 optimal solutions in sets with 30, 60 and 90 jobs, respectively. These results would be remarkable if there are not so many groups with very large gaps. Some gaps are larger than 100%, which indicates that the optimization could take too much time to finish.

Since the mathematical formulation lacks performance as the instances become bigger. In the second experiment, the algorithm ACO was applied on all instances using the parameters value in Table 2 and a time limit of 20, 40, 80 and 160 seconds for projects with 30, 60, 90 and 120 jobs, respectively. ACO is a stochastic method, so 20 independent executions were done. The ACO parameter calibration was performed with a set of 50 randomly chosen instances. The parameters values tested were: $nAnts$=100, 200 or 300; p_0=10.0, 20.0 or 50.0;

maxPheromone=2.0, 4.0, 8.0 or 16.0; *dissipRate*=0.1, 0.2 or 0.3. Table 5 classifies the best solution found by its relative difference with the CPLEX bounds. Column "Gap DB(%)" indicates the average distance to that dual bound. Column "Improv." shows how many times the ACO produced a better solution than CPLEX. The average distance to CPLEX final incumbent solution is also showed in the last column.

Table 5. Comparisons between ACO solution and CPLEX bounds through relative difference intervals. Only projects with 30 jobs.

	Relative distance to the dual CPLEX bound						Incumbent Solution	
Groups	0%	(0%,2%]	(2%,5%]	(5%,10%]	(10%,∞)	Gap DB%	Improv.	Gap Sol.%
1-4	12	17	10	1	0	1.2	0	1.2
5-8	6	13	8	7	6	4.6	2	1.7
9-12	1	11	10	5	13	8.5	4	1.5
13-16	4	8	6	9	13	10.4	8	0.8
17-20	14	20	6	0	0	0.8	0	0.8
21-24	8	10	12	6	4	3.7	1	1.7
25-28	6	6	12	6	10	8.5	8	1.1
29-32	4	10	3	5	18	12.8	9	-0.2
33-36	17	21	2	0	0	0.5	0	0.5
37-40	14	16	3	3	4	2.8	2	0.6
41-44	6	6	10	7	11	9.3	8	0.4
45-48	5	8	5	8	14	12.1	6	0.5
Total	97	146	87	57	93	-	48	-

Most of the ACO solutions (330 solutions, 97 of them are optimal ones) are up to 5% worse than CPLEX dual bound. Groups 13-16, 29-32 and 45-48 have the hardest instances, where the CPLEX dual bound was almost not improved and the ACO solution were relatively distant from that bound. This result does not indicate that ACO provided poor solutions to those sets of instances. Actually, in such instances, the ACO found 48 solutions better than CPLEX incumbent solution. In groups 29-32, the ACO solutions (time limit of 20 seconds) were 0.2% better than CPLEX incumbent solution (time limit of 1800 seconds), on average. In such instances, the MIP formulation had a worse performance.

Finally, in Table 6, the average efficiency achieved by ACO is compared to some data provided by PSPLIB-energy [16]. The PSPLIB-energy gives the average results of all instances with the same size. Besides that, no further information is given about stopping criteria or computational environment used to obtain those results. Although a deeper comparison is not possible, the average efficiency obtained by ACO is competitive to PSPLIB-energy data (approximately, 2% better).

Table 6. Average efficiency of ACO and literature data (higher values are better).

Jobs	30	60	90	120
ACO	64.95	66.66	67.57	51.82
Library	62.93	64.24	64.78	50.32

5 Concluding remarks

This paper dealt with a new variant of the classical RCPSP. In this variant, called MRCPSP-energy, each job has different execution modes where duration and energy consumption are conflicting. This trade-off is reflected on the problem objective, which is maximizing the total project efficiency. In other words, it is necessary to reduce the project makespan while decreasing the total energy spent simultaneously.

Since there are no specific methods for MRCPSP-energy in the literature, a Mixed Integer Programming formulation and an Ant Colony Optimization metaheuristic were proposed. The exact approach was only used in small instances, but provided significant results. CPLEX gaps were not too large, albeit for some instances the optimization struggled. The proposed ACO found competitive solutions compared to CPLEX bounds and to the few available solution data in the literature.

As future works, a local search method could improve even more the ACO performance. The creation of other tightening constraints may diminish the gap achieved by the MIP formulation.

References

1. J. Blazewicz, J. Lenstra, A. Rinnooy Kan Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.
2. F.F. Boctor. Some efficient multi-heuristic procedures for resource constrained project scheduling. *European Journal of Operational Research*, 49:3–13,1990.
3. K. Bouleimen and H. Lecocq A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149:268–281,2003.
4. P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1999.
5. N. Damaka, B. Jarbouia, P. Siarryb, T.Loukila. Differential evolution for solving multi-mode resource-constrained project scheduling problems. *Computers & Operations Research*, 36:2653–2659,2009.
6. D. Debels, B. De Reyck, R. Leus, M. Vanhoucke. A hybrid scatter searchelectromagnetism metaheuristic for project scheduling. *European Journal of Operational Research*, 169:638–653,2006.
7. E. Demeulemeester, W. Herroelen. A branch-and-bound procedure for multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818, 1992.

8. A.M. Fahmy, T.M. Hassan, H. Bassioni Improving RCPSP solutions quality with Stacking Justification – Application with particle swarm optimization. *Expert Systems with Applications*, 41(1), 5870–5881, 2014.
9. J.F. Gonçalves, J.J.M. Mendes, M.G.C. Resende A random key based genetic algorithm for the resource constrained project scheduling problems. *International Journal of Production Research*, 36:92–109, 2009.
10. S. Hartmann, D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1), 1–14, 2010.
11. R. Kolisch. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management* ,14:179–192, 1996.
12. T. Liao, T. Stützle, M. A. Montes de Oca, M. Dorigo. A unified ant colony optimization algorithm for continuous optimization. *European Journal of Operational Research*, 234(3):597–609, 2014.
13. S. Liu, C. Wang Resource-constrained construction project scheduling model for profit maximization considering cash flow. *Automation in Construction*, 17:966–974,2008.
14. A. Mingozi, V. Maniezzo, S. Ricciardelli, L. Bianco. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, 44:714–729, 1998.
15. P. B. Myszkowski, M. E. Skowroński, Ł P. Olech, K.Oślizło", Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem. *Soft Computing*, 19(12):3599–3619,2015.
16. D. Morillo PSPLIB-ENERGY: a PSPLIB extension for evaluating energy optimization in MRCPSP. *http://gps.webs.upv.es/psplib-energy/*, last access on July, 25th, 2017.
17. K. Nonobe, T. Ibaraki. Formulation and Tabu search algorithm for the resource constrained project scheduling problem (RCPSP). *In: Ribeiro, C.C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics.* Operations ResearchComputer Science Interfaces Series, Kluwer Academic Publishers 15, 557–588, 2002.
18. D. M. Torres, F. Barber, M. A. Salido MRCPSP-ENERGY, un enfoque metaheurístico para problemas de programación de actividades basados en el uso de energía. *Proceedings of XVIII Latin Ibero-American Conference on Operations Research (CLAIO XVIII)*, SantiagoChile, October, 2016.
19. L-Y Tseng, S-C Chen A hybrid metaheuristic for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 175(2):707–721,2006.
20. V. Valls, F. Ballestin, M.S. Quintanilla. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 185(2), 495–508, 2008;
21. G. Zhu, J. Bard, G. Tu A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem", *Journal on Computing*, 18(3):377–390,2006.