Pesquisa Operacional (2014) 34(1): 125-142

Printed version ISSN 0101-7438 / Online version ISSN 1678-5142
www.scielo.br/pope

A HYBRID METHOD FOR A PROJECT SCHEDULING PROBLEM

André Renato Villela da Silva

Received February 20, 2013 / Accepted October 22, 2013

ABSTRACT. This work deals with a project scheduling problem where the tasks consume resources to be activated, but start to produce them after that. This problem is known as Dynamic Resource-Constrained Project Scheduling Problem (DRCPSP). Three methods were proposed to divide the problem into smaller parts and solve them separately. Each partial solution is obtained by CPLEX optimizer and is used to generate more complete partial solutions. The obtained results show that this hybrid method works very well.

Keywords: Project Scheduling Problem, Combinatorial Optimization, Hybrid Methods.

1 INTRODUCTION

In Project Scheduling Problems (PSP), two elements are essential: the tasks that represent each step of the project to be executed, and the resources that are necessary inputs for a task to be performed. The tasks are linked together via precedence relationships that determine the order in which tasks may or may not be performed. Typically, these relations are called finish-to-start, i.e., a task must be completely executed before a successor task begins. It is also very common that a task may have more than one predecessor. In this case, all predecessors must have been performed before starting the task at hand. The most common goal is to make all project tasks to be executed as soon as possible, respecting the precedence constraints and resource availability.

The resources that are necessary to perform the tasks are the other element to be treated in PSPs. A fairly traditional classification divides them into two groups: renewable and non-renewable resources. Renewable resources are those which, after being used in the execution of a task, are again available to be used by another task not yet executed. Some examples of this class are machines (excavators, tractors, computers) and professionals (engineers, programmers, assistants). These resources can be reused at the end of a project stage. Resources are classified as non-renewable if they are available only once during the whole planning horizon in which the problem should be treated. Once they are used (consumed) we can no longer count on them until the end of the problem. Common examples are money and fuel, among others.

Universidade Federal Fluminense, Instituto de Ciência e Tecnologia (PURO), Departamento de Computação, R. Recife s/n., 28895-532 Rio das Ostras, RJ, Brazil. E-mails: avillela@ic.uff.br; andre.renato@gmail.com

Traditionally, PSPs do not suppose generation of resources but only consumption of them. They are input data and have initial values defined *a priori* since either they are non-renewable ones or they have a renewal rate very well defined by the problem, as seen in [17, 25]. These scenarios, however, are unable to model certain situations where, from the end of the task execution, it starts to generate additional resources.

In order to illustrate these situations, suppose that the project in question is the commercial expansion of a company. After the construction of a new branch, it is reasonable to assume that the branch can financially contribute to the matrix through the branch profit that is expected. The most important resource in this case is undoubtedly the financial return. The amount once invested in building branches does not become available again at the end of this step, like a machine or a worker. What happens is that, after completion of this investment, the branch starts to produce its own financial resources that can be applied in the execution of other steps: opening of new branches, hiring staff, purchasing equipment, among others.

An example of this situation can be seen at the timeline of the Brazilian company Net [16]. Nowadays, Net provides several communication services such as cable TV, internet and telephony over many Brazilian cities. However, in 1991, Net started providing only TV services for 100 customers in Campo Grande/MS city. Two years later, the company bought other small TV companies reaching almost 5000 clients and spreading its business to neighbor estates. In 1997, Net´s budget was enough to acquire some more companies and expand its cable network over several important cities, with a valuable client base. After many other acquisitions, in 2007 Net had more than 2 million clients and an annual growing rate of 16%. In the same year, the annual growing rate of telephony services was 212%. The company continues to invest on its base client expansion in country cities or in far capital cities. It is important to say that the strategy used by Net was to expand first over cities with higher expected profits in order to accumulate more resources and grow faster.

Obviously, the business world is far more complicated than the model studied in this paper, but some concepts explained later are present in real-life situations as cited above. Precedence constraints are not always easy to model. For infra-structure companies is not hard to imagine that the expansion can not be randomly done due to physical aspects and high costs related. For store-based retailing companies such precedence constraints are much less rigid. This paper deals with a theoretical model that can be used in a general way, so precedence constraints are present in this model. The problem studied in this paper finds potential applications in commercial or industrial expansions projects.

In the model that will be the subject of study, money is a renewable resource, but not like in traditional models, where there is a maximum amount available to each unit of time. In the proposed model, a resource is renewable because it is possible to have a reduction and increase of its available quantity over time. The production of resources (money in this case) remains from the completion of a task until the end of the planning horizon in question.

This paper will address the PSP where tasks consume resources when activated and, thereafter, begin to generate resources until the end of the planning horizon whose size is given by an

input parameter. The model also assumes an initial amount of resources that can be spent in the first activations. The objective of this model is reaching the end of the planning horizon with the greatest possible amount of resources. The resource discussed in this case may not have a defined maximum, since the objective would lose its meaning. This resource that is consumed and then produced varies in quantity over the planning horizon. Therefore, this type of resource is called Dynamic Resource, and the scheduling problem called Dynamic Resource-Constrained Project Scheduling Problem - DRCPSP.

The remainder of this paper is organized as follows: Section 2 presents a brief literature review of the problem. Section 3 will bring the formal definition of the problem. Hybrid methods – that mix characteristics of exact algorithms and heuristics – will be the subject of Section 4. In Section 5, the computational results are presented. Finally, the conclusions are presented in Section 6.

2 LITERATURE REVIEW

Resource-constrained Project Scheduling Problems (RCPSP) are generalizations of Task Scheduling Problem, where we have a set of tasks which must be assigned to processing units such that the last task finishes as soon as possible. The time when it occurs is called makespam. RCPSP is a NP-hard problem, being on of the most studied problems of this class [26]. In resource-constrained versions [11], minimizing the makespam is still the primary objective. However, a new component called resource is added to the problem definition.

Resources can be understood as anything needed to a task in order to be scheduled. We can mention raw materials, worker or equipments as resources examples. Each task has an amount of required resources that must be available when the task is activated. Some models accept many kind of different resources and many ways of activating the same tasks according to the amount of resources [5].

Resources can be classified as non-renewable when, once it is used, it cannot be recovered or re-utilized after the task execution. Renewable resources become available again after their utilization by the tasks. Usually, the problem specifies the total amount of each kind of resource that is available at the problem beginning. There is another kind of resource called partially renewable resources [2], which works as renewable resources at some time units and as non-renewable resources at the other time units.

Many approaches have been used to tackle RCPSP. First techniques used only constructive heuristics [3, 12, 15] or mathematical formulations [6, 14]. However, over the last years, many metaheuristics approaches have obtained significant results in several areas [1, 8, 17, 19, 20, 24, 25], including validation and comparison with third-party software like [10]. They are particularly applied to the multi-mode version of RCPSP, where a task can be executed in different ways, using different amount of resources [4, 5]. For the standard version of RCPSP, one of the most effective metaheuristic can be seen in [9].

An early work on the DRCPSP was introduced by [21]. The work objective was to propose a solution to the main weakness of many Evolutionary Algorithms (EAs): premature convergence.

After a few generations, previously proposed EAs had difficulty to keep improving the quality of individuals through the crossover operator. The article applied a GRASP version [7] capable of a complete discarding of those solutions with poor quality or that could not be improved in efficient way.

In GRASP, at each iteration, a new solution is generated by a constructive method and then improved by a local search. A new local search proposed in that article was tested, besides new tasks choosing criteria and a new local search called LS3. It starts from a full schedule already built and uses a criterion less greedy than that used by the constructive method.

In order to provide a good result, the LS3 needed that the fixed partial solution was neither short, because few information could be taken from the provided solution nor too large, because in that way there would be no room for significant modifications. The computational results shown in that article indicate that the search LS3 is capable of improving a solution much better than the previously proposed local searches. The two GRASP versions tested by that work performed better than the previously proposed EAs.

Local search LS3 was reemployed in [22] as part of a new evolutionary algorithm. This time, in addition to the basic evolutionary operators, other mechanisms have been proposed trying to avoid the premature convergence. The mechanisms of intensification and diversification start acting when the population passes a few generations without improving the best solution found. In the first place, the best solution so far will serve as a seed for the generation of quite similar individuals. Using the search LS3, this best individual leads an entire population of new others. This phase acts as an intensification around this good solution, which might make better solutions to be more easily found.

If however these solutions are not found and a few more generations follow without improvements on the best solution, the entire population will be discarded and a new one is created from the constructive algorithm ADDR, as was done in the initial population of the evolutionary algorithm. This phase aims to diversify the focus of the EA to solutions other than that it is currently working on.

Also, in this paper, a hybrid method able to combine a partial scheduling generated by an exact method with the proposed evolutionary method was proposed. This partial scheduling is achieved by CPLEX software, according to the mathematical formulation proposed in [21] and constraining the activation of tasks until a given time unit. The goal is to generate the populations from both partial scheduling and local search LS3. With the first part of the schedule given to the exact method, it is expected to delivery to the local search a partial solution of superior quality compared to those generated in a heuristic way. The hybrid method has proved to be particularly effective in instances that had a relatively short planning horizon (up to approximately 25 units of time), even if there was a large number of tasks. With longer planning horizons, the computational time spent by the first part of the schedule was not compatible with the quality of the solution generated, which indicated that this method was not good for that kind of instances.

Further studies such as [13, 18] show that methods that make good use of features from more than one type of heuristics or that mix heuristics and exact components can provide very interesting results which could hardly be achieved by traditional heuristic methods.

3 PROBLEM DEFINITION

Before presenting the definition of the problem, it is important to clarify some concepts used in the model. Some of them come from the classic Project Scheduling Problems, others were defined specifically for this model. Concepts marked with * represent an instance input data.

- *Task**: is each stage of a project that should be performed. Its most important data is the time unit when it should be executed.
- *Solution*: is a vector of n non-negative integers, where n is the number of project tasks. Each element v_i of the vector indicates the time when the task i must be executed.
- *Resource*: any input (material, equipment, professional or any other item) required to perform each of the project tasks. Although possible in several other models, the DRCPSP admits only a single type of resource that can accumulate non-limited quantity of units throughout the planning horizon.
- *Available Resources*: are the resources that can be applied for executing tasks in a given time t. It is denoted by Q_t.
- *Cost** of a task: is the amount of resources (positive value) needed for the execution of a task. The cost of a task i is denoted by c_i.
- *Profit** of a task: the non-negative quantity of resources that will be produced by the task at each time unit, starting from the moment of time following its activation until the end of the planning horizon. It will be denoted by p_i for each task i of the problem.
- *Accumulated Profit*: is the sum of the activated tasks profit at a time unit t. The notation is made by P_t and P_0 is usually equal to zero.
- *Activation time*: is an indication of a time unit t at which the task should be executed. At this point in time, available resources must be equal to or greater than the cost of the task in question. It is also necessary that all predecessor tasks are already activated until the previous time unit $(t - 1)$.
- *Task duration**: is the time required for a task to be fully processed, i.e. have completed their execution. In DRCPSP, by the end of the time unit of its activation the task is considered completed and then starts to produce resources until the end of the planning horizon.
- *Planning horizon**: is the time interval at which tasks can be executed. In DRCPSP it is discretized in time units from 1 up to H, which is a problem input.

The DRCPSP is composed of a directed acyclic graph $G = (V, A)$, where V is a set of vertices and A is the set of arcs connecting the vertices. Each task $i \in V$, is associated with a cost c_i and a profit p_i, non-negative integers. Initially, there is an amount of available resources

$Q_0 > 0$ and an accumulated profit $P_0 = 0$. The scheduling should be conducted over a planning horizon consisting of H time units. The problem objective is to maximize the amount of resources (available resources and accumulated profit) at the end of the planning horizon.

Figure 1 and Table 1 show an example of this scheduling model solved by an arbitrary algorithm. In the example, the planning horizon is composed of four units ($H = 4$) and the initial amount of available resources $Q_0 = 4$. For the sake of simplification, the amount of available resources Q_t and accumulated profit P_t will be denoted by Q and P, respectively. The already activated task are in white, available task are in gray and tasks that can not yet be activated are shown in black.

Table 1 – Costs and profits used by scheduling example.

Task	Cost	Profit
1	2	1
2	3	2
3	4	4
4	1	2
5	2	3
6	4	5

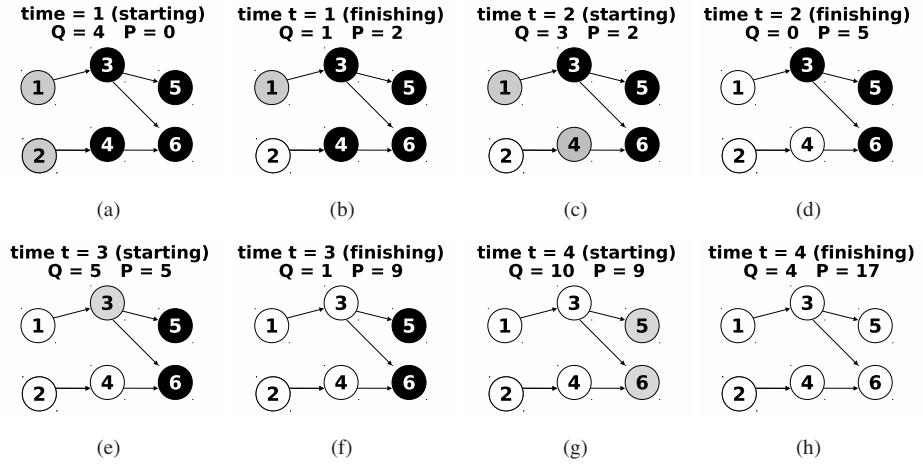

Figure 1 – Steps of scheduling example.

At time unit $t = 1$ (Figure 1), tasks 1 and 2 are available for activation. Suppose the scheduling algorithm chooses task 2 to be activated. It is necessary to consume 3 units of resource, since $c_2 = 3$. The accumulated profit which initially was zero now contains 2 units ($p_2 = 2$). It is not possible to activate task 1 with the remaining resources, so, time unit $t = 1$ must be finished like in Figure 1. When the scheduling algorithm moves to the next time unit ($t = 2$, Figure 1(c)), the

accumulated profit turns into available resource, since the profit represents the resources that are being produced by the task already activated. With the activation of task 2 performed, task 4 is able to be activated.

The amount of available resources now allows that both task 1 and 4 may be activated. This decision subtracts 3 units of resources ($c_1 + c_4 = 3$) and adds 3 units ($p_1 + p_4 = 3$) to the accumulated profit, like in Figure 1(d). As there are no more available task right now, it is necessary to advance to the next time unit. The accumulated profit is added to the available resources and tasks state is updated.

In the beginning of time $t = 3$ (Figure 1(e)), task 3 can be activated because there are enough available resources. After updating available resources, accumulated profit and tasks state, the algorithm reaches the last time unit ($t = 4$, Figure 1(g)) when tasks 5 and 6 will be activated. At the end of this time unit (Figure 1(h)), there are 4 available resources and the accumulated profit is 17. The result of this scheduling process is equal to 21 units of resources.

3.1 Mathematical formulation

In [23] an improved mathematical formulation was proposed for DRCPSP. In that formulation, called $F2$, variables y_{it} indicate whether a task i was activated at time t or before that. A zero value means that a task i was still not activated and an one value means that a task i was already activated at that time t. So, when a task is activated at time t, all variables related to that task, from time t to time H, have to be fixed at value one. Since the problem allows only one activation at most, it is possible to know if a task was activated earlier looking at only one variable. The formulation is presented and described next.

(F2) Max $\quad (Q_H + P_H)$

Subject to

$$y_{it} \leq y_{it+1} \quad \forall i = 1, \ldots, n \quad \forall t = 1, \ldots, H-1 \tag{1}$$

$$y_{it} \leq y_{jt-1} \quad \forall i = 1, \ldots, n \quad \forall t = 2, \ldots, H \quad \forall j \in Pred(i) \tag{2}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i (y_{it} - y_{it-1}) \quad \forall t = 1, \ldots, H \tag{3}$$

$$P_t = \sum_{i=1}^{n} p_i y_{it} \quad \forall t = 0, \ldots, H \tag{4}$$

$$y_{i0} = 0 \quad \forall i = 1, \ldots, n \tag{5}$$

$$y_{it} \in \{0, 1\} \quad \forall i = 1, \ldots, n \quad \forall t = 1, \ldots, H \tag{6}$$

$$Q_t, P_t \in N \quad \forall t = 0, \ldots, H \tag{7}$$

Constraints (2) ensure that a task i, after its activation time t, remains activated until the end of the planning horizon. Constraints (3) model precedence between a task i and each predecessor task j. Constraints (4) define how available resources Q_t are computed over time. The amount

of available resources Q_t is equal to the remaining resources from previous time Q_{t-1} plus all profits produced at that time P_{t-1}. Constraints (5) indicate the accumulated profit P_t at a time t being the profit sum of all activated tasks. Constraints (6) show that no tasks are activated at the beginning of the planning horizon and constraints (7) and (8) define y_{it} as binary variables, Q_t and P_t as non-negative integer variables.

4 PROPOSED METHODS

The hybrid algorithm CPLEX + EA3 proposed in [22] introduced a very interesting idea: to divide the planning horizon into two halves and perform the scheduling of these parts in sequence, using CPLEX and the evolutionary algorithm EA3, respectively. The algorithm presented good results for instances with a short planning horizon or a relatively small number of tasks (up to 300 tasks).

For larger instances, especially those whose planning horizon was extensive, the algorithm spent too much time to produce the first half of the partial scheduling (under the responsibility of CPLEX) and, consequently, could not terminate in an acceptable time.

The proposal now is to use the best formulation known for the DRCPSP, originally discussed in [23], and divide the planning horizon into smaller parts so that a partial result is produced faster. This solution is then used as starting point for another partial solution (covering more units of time) to be produced until the entire planning horizon is covered. Additional mathematical constraints that prevent activation after the desired time unit are incorporated to the model. Whenever a partial solution is produced, the already activated tasks have their variables fixed into the formulation according to the activation time of the task. This must be done so that the problem does not become increasingly difficult with the possibility of scheduling in increasing intervals.

The division of the planning horizon can be done in several ways. Three proposals have been done using two different criteria that will be explained below. As both criteria do not treat the entire planning horizon at once, proposed methods are considered heuristics and may not provide the optimal solution to the problem. Each partition generated, however, will be solved by an exact method, allowing to classify the proposed strategies as hybrid methods.

- Fixed-length intervals: in this way of division, the planning horizon will be sectioned into intervals of K time units. For example, if $H = 7$ and $K = 3$, there will be 2 intervals whose length will be 2 and the last interval will have length of 1 time unit.

- Variable-length intervals: another way to accomplish the division is to define the amount of intervals but allow them to have different lengths. Although the ideal is that the lengths are similar, the major impact on the problem hardness is caused by the amount of tasks (and consequently of binary variables) to be treated in each interval. Thus, intervals whose lengths have a small difference are perfectly acceptable.

  However, preliminary tests showed that taking into account only the amount of binary variables did not generate so good results because the first time units had few binary variables

due to precedence constraints. This leads to early intervals that need to incorporate more units of time, taking longer to run. In fact, the factor that will determine the length of each interval should be based on the amount of tasks that have the earliest activation time within the interval in question.

A preprocessing algorithm computes at each time t, the number of tasks i that have $earliest(i) = t$, this quantity will be denoted by $w(t)$. The value of n (the total number of tasks) will be divided by the desired number of intervals v to know the intervals average length m. The intervals aggregate consecutive time units $t_i, t_{i+1}, \ldots, t_{j-1}, t_j$ until $\sum w(t)_{t=t_i}^{t_j}$ to be the closest to m as possible. If the sum can not reach the exact value of m, the amount left over or missing will be taken into account for the next interval. It is important to mention that the amount of intervals v is a input data given by the user of the algorithm. Figures 2 and 3 show how to divide a graph into four or six intervals, respectively, in which tasks may be started until the time $t = 8$. No arcs were shown in order to ease the visualization of the figures. The tasks are represented by gray circles.

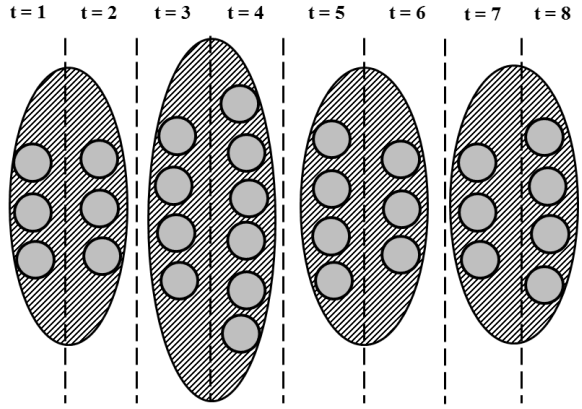

Figure 2 – Example of a planning horizon divided into 4 partitions.

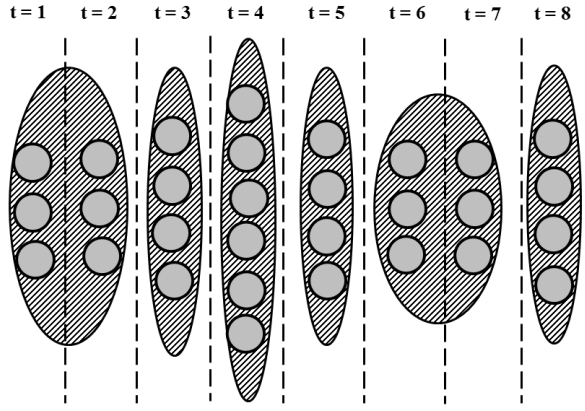

Figure 3 – Example of a planning horizon divided into 6 partitions.

As the graph contains 30 tasks, the division into four intervals in Figure 2 gives us an average of 7.5 tasks per interval. The first time unit has only 3 tasks, which is too far from our goal (7.5). Including the second time, we get 6 tasks. If we include time 3, we would have 10 tasks – farther from 7.5 than the previous value of 6. So our first interval ends at $t = 2$, missing a task and a half to the goal. The second interval will take this into consideration and look for a group that comes close to $7.5 + 1.5 = 9$ tasks. The best choice is to group time units 3 and 4 into another interval, now with a task to spare. The algorithm continues until it reaches $t = 8$ in this case.

For division into six intervals we have an average value of five tasks for each partition. In Figure 3 it is possible to see how these partitions would be employing the same algorithm.

- Multiple intervals of variable length: the idea is quite similar to the previous one. An original configuration of interval lengths is computed using the same criteria. After this, a new configuration is obtained by shortening the first interval with one time unit given to the next interval. Consequently, the next interval is enlarged by one time unit. All other intervals remain fixed. Another configuration is given by enlarging the first interval from the original configuration retrieving this time unit from the second one. The same process of shortening and enlarging a interval based on the original configuration is done for each interval. The scheduling algorithm used by previous partition method is executed for each produced configuration. The best result from all of these configurations is the solution provided by this third partition method. Figure 4 shows an example with 5 intervals ($v = 5$) and $H = 24$. Each time unit is represented by a small square and the intervals by a vertical dashed line. Squares that present the same background style are in the same interval.

These three ways of partitioning can make the optimal solution unreachable, because the optimal solution of a partial schedule may not be part of the optimal solution from the whole schedule. Thus, these hybrid versions can not be considered exact algorithms for the DRCPSP.

5 COMPUTATIONAL RESULTS

First, it is necessary to address the instances used in the computational experiments. All tests were made with an Intel Quad-core Q9550 with 8 GB of RAM, running Linux (Ubuntu 8.10) an CPLEX 11.2 (parallel version up to 4 threads). Instances have been retrieved from LABIC project (`http://www.ic.uff.br/~labic`) and all have non-trivial solutions.

Graphs are composed of 10% of tasks without precedence, other tasks have up to 5 randomly chosen predecessors. Costs and profits are randomly chosen in the range [1;50] and [1;10], respectively, in order to produce interesting instances. Higher costs would make only few tasks to be activated; lower values would provide very easy instances where all tasks could be activated. Planning horizon size was defined as square root of n (number of tasks) and the initial amount of resources Q_0 is enough to activate at least one task without precedence. P_0 is always equal to zero. These instances have been used by all papers about DRCPSP. Instances from other project scheduling problems are not suitable for the DRCPSP because they do not have profit

Figure 4 – Multiple intervals scheme.

values. Introducing artificial values in those instances and using them to compare the proposed formulation are not the objective of this paper.

5.1 Computational experiments

The experiments with the hybrid methods consist in executing the algorithm until all time units are covered. Although the division of the planning horizon abbreviates the time required to complete the optimization in each interval, it is not possible to predict a maximum time for this to occur. Thus, a limit of 3 hours (10800 seconds) will be given for the optimization of each partition. If the limit is reached, the incumbent solution is considered as the result of this interval.

The first scheme performs the division at intervals of size K. Table 2 shows the final results for several values of K (2 to 10). The last column shows the results average. The instances are the same as those analyzed by [21].

Two subdivisions with similar values of K generated close results. It was expected that a larger value of K produced better results by covering a larger portion of the planning horizon. This was not confirmed because the view of only part of the scheduling may influence the algorithm on how much important a task is for the whole scheduling. Example: A task can be considered not profitable if we analyze it alone and do not consider the activating of its successors. However, if there is enough time for the successors activation, the same task may be considered very important because its successors could produce much resources.

Table 2 – Results for planning horizon partitioning with fixed-length intervals.

Instance	Interval Length (K)									Average
	2	3	4	5	6	7	8	9	10	
100a	303	304	303	**304**	**304**	**304**	**304**	**304**	**304**	303.8
200a	632	628	632	**636**	632	635	632	635	**636**	632.5
300a	1898	1927	1965	1946	2020	2041	**2043**	2025	1950	1959.2
400a	5618	6143	6641	6169	**6741**	6732	6656	6460	6158	6320.9
500a	12689	13239	13006	13433	13397	13408	**13941**	13846	13510	13272.0
600a	8890	9554	9424	9321	9809	9526	9417	9864	**10096**	9407.9
700a	25454	28769	28398	28599	28772	29882	**30587**	29919	28824	28209.5
800a	32889	35127	35614	34118	35683	35300	37007	**37363**	37145	35263.0
900a	31981	33635	35070	33594	36046	34442	34619	36730	**37529**	34372.9
1000a	64744	65840	65207	67699	67885	66024	67867	69338	**69717**	66473.3

Obviously a very big difference between two values of K tends to produce different results. Note that the time spent by each partition scheme does not have an increasing behavior. In fact, each value of K corresponds to a new set of partial schedules, whose practical difficulty is too complicated to be predicted. Table 3 shows the total time spent (in seconds) for each value of K.

Table 3 – Computational time (secs) for the planning horizon partitioning in fixed-length intervals.

Instance	Interval Length(K)									Average
	2	3	4	5	6	7	8	9	10	
100a	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.1
200a	0.0	0.2	0.2	0.3	0.3	0.3	0.4	0.7	0.9	0.3
300a	0.9	0.6	0.7	1.0	1.4	2.6	0.9	1.4	1.8	1.3
400a	1.4	0.9	0.9	2.4	1.8	2.0	3.8	14.1	43.1	7.3
500a	2.5	2.9	5.6	2.6	12.1	9.0	10.5	41.5	2043.7	213.5
600a	3.8	2.9	3.4	11.3	12.7	7.5	514.3	870.5	50.0	148.3
700a	5.3	5.8	5.2	8.5	11.7	31.3	23.0	19.6	22.3	14.4
800a	7.5	7.4	16.0	9.2	203.7	164.1	1346.3	2081.9	555.1	440.6
900a	9.3	7.7	7.8	22.4	59.9	10805.9	10803.9	7788.6	1729.8	3125.2
1000a	13.0	9.3	12.3	9.7	1158.3	102.1	694.1	22.2	1742.6	378.7

5.2 Variable-length intervals

In this experiment, the partitioning of the planning horizon takes into account the number of tasks that can start activation in each interval. Thus, it is possible to have some partitions with fewer time units and other partitions with a little more. However, the amount of tasks should be as similar as possible. Table 4 shows the final result for the planning horizon partition into 2 to 10 intervals ($v = 2, \ldots, 10$).

The results of this second strategy does not have a well-defined behavior related to the amount of intervals, similarly to the division into fixed-length intervals. In theory, the smaller the amount of intervals, the larger they would be and better should be the solutions. However, to determine a perfect mapping between Tables 2 and 4 is very complicated because even in an instance where a value of K corresponds to v intervals, these intervals can group tasks completely different. Suppose a instance with 400 tasks and a planning horizon with $H = 20$. The first division using $K = 10$ take two steps: $H_1 = 1 \ldots 10$ and $H_2 = 11 \ldots 20$. When the partitioning is done by the second criterion with $v = 2$, we can have two steps $H_1 = 1 \ldots 10$ and $H_2 = 11 \ldots 20$, or $H_1 = 1 \ldots 9$ and $H_2 = 10 \ldots 20$, or $H_1 = 1 \ldots 11$ and $H_2 = 12 \ldots 20$ or any other depending on the amount tasks that are included in each interval. Therefore it is not possible to make a comparison case-by-case, but if taken the results of both criteria together with the computational times shown in Tables 3 and 5, it can be seen that the second criterion is slightly more efficient.

Table 4 – Computational results for the planning horizon partitioned with variable-length intervals.

Instance	Amount of intervals (v)									Average
	10	9	8	7	6	5	4	3	2	
100a	**304**	**304**	**304**	**304**	**304**	**304**	303	**304**	303	303.8
200a	627	628	628	628	628	628	629	631	**635**	629.1
300a	1784	1833	1869	1839	1885	1841	2026	2030	**2042**	1905.4
400a	5374	5999	5570	5503	5503	5647	**6641**	6612	6460	5923.2
500a	12994	13113	13376	13288	13230	13501	13846	13938	**14225**	13501.2
600a	9183	9110	9230	9441	9414	9814	9830	**10090**	10035	9571.9
700a	25106	25237	25135	25291	27415	29311	27445	28563	**31329**	27203.6
800a	32686	33542	34555	35287	36362	35601	37508	37089	**38511**	35682.3
900a	32447	32004	31705	34266	34345	34431	36475	37295	**38684**	34628.0
1000a	63840	63796	64821	62294	64253	65332	65779	68297	**70519**	65436.8

Take for comparison the case where the second criterion is applied with $v = 2$. Theoretically, this is the hardest partitioning because it has only two large intervals. Only the whole planning horizon at once is more difficult than this scheme. All instances with over 400 tasks in these experiments have more than 20 units of time. Using the first partition scheme and $K = 10$, we have at least three partitions, the last one being usually smaller than the others. The total computational time spent for these instances, by the first criterion was 29184.3 seconds. The second criterion took only 25442.0 seconds. The results of the second criterion is 4.5% better than those from the first criterion. Only in one instance (600a) the first criterion had better result. If we consider also the instance 400a, which was divided into two partitions by the two criteria, the second one would perform even slightly better. Thus, the second criterion with only two (larger) intervals performed better than the first criterion with three or more (smaller) intervals.

The partitioning by variable-length intervals has, as major advantage, the fact that the first intervals include a greater number of tasks than the partitioning by fixed-size intervals. At the first levels of the graph few tasks are available to be activated. Thus, to achieve the average number of tasks per interval, it is necessary to incorporate more time units. With more time units initially

being considered the better will be the partial scheduling at those intervals. As further schedules are based on results of previous schedules, the final result tends to be better.

All of these aspects that make sense in theory proved to be good on the practice too. The smaller time consumption can be explained similarly: initial schedulings do not usually take long. If they were well made, it tends to offer better primal bounds for the following schedulings, finishing faster.

Table 5 – Computational times for the planning horizon partitioned with variable-length intervals.

Instance	Amount of intervals (v)									Average
	10	9	8	7	6	5	4	3	2	
100a	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0
200a	0.0	0.0	0.0	0.0	0.1	0.4	0.2	0.2	0.5	0.2
300a	1.0	1.0	0.8	0.8	0.6	0.6	1.0	1.5	5.7	1.4
400a	1.2	1.0	1.2	1.1	1.0	0.8	0.9	1.3	16.9	2.8
500a	3.2	2.8	2.5	3.8	2.2	3.2	3.3	11.0	342.5	41.6
600a	3.5	3.2	5.2	3.5	3.9	8.8	20.3	172.4	9910.8	1125.7
700a	4.4	4.2	3.7	3.3	3.3	3.2	3.4	7.1	164.7	21.9
800a	6.0	5.6	5.5	5.3	18.2	5.6	29.6	87.4	6616.0	753.2
900a	8.6	6.4	6.3	5.9	7.0	5.4	28.1	81.5	4072.4	469.1
1000a	8.5	8.3	7.7	6.9	7.1	6.9	7.8	31.8	488.1	63.7

5.3 Multiple intervals

This third partitioning criterion can be seen as an extension of the previous one, since successive executions are done addressing the "neighborhood" of the original configuration. In fact, the number of executions is equal to $2v-1$, where v is the desired number of intervals. It is important to remark that the last interval must always finish with $t = H$. Table 6 shows the results for several values of v.

As expected, the results of this partitioning strategy are a little better than before – around 2.8%. However, the computational time is much higher, not only because many iterations are performed, but also because when the length of the intervals is changed, they become more difficult to solve. Table 7 shows the execution times. According to the presented results, it can be said that this method has a cost-benefit ratio worse than the previous one, since it takes much longer even when there are large numbers of intervals. Therefore among the three partitioning strategies, the second one – with intervals of variable length – presented the best results in a time not too long.

Finally, Table 8 brings an interesting comparison between the best result obtained by partitioning against the primal bound obtained by CPLEX, assuming a maximum of 50000.0 seconds as explained in [23]. As already mentioned, the partitioning of the planning horizon does not treat the whole problem at once. This implies in not necessarily to achieve the global optimum. Although the partitioning algorithm can be considered a hybrid method, its final outcome shows a stronger

Table 6 – Results for multiple intervals with variable length.

Instance	Amount of intervals (v)									Average
	10	9	8	7	6	5	4	3	2	
100a	304	304	304	304	304	304	304	304	304	304.0
200a	**636**	632	632	632	632	632	635	**636**	635	633.6
300a	1830	1890	1901	1929	2003	1993	2026	2053	**2061**	1965.1
400a	6311	6349	6156	6349	6349	6510	6716	**6741**	6656	6459.7
500a	13441	13273	13490	13501	13369	13792	14014	14127	**14239**	13694.0
600a	9429	9429	9342	9616	9702	9969	9921	10123	**10167**	9744.2
700a	27321	28272	27297	28844	28830	29310	28409	29810	**31567**	28851.1
800a	35877	35432	35878	36140	37456	35663	37692	38096	**38579**	36757.0
900a	33144	33678	35034	34982	35953	36404	37537	38287	**38793**	35979.1
1000a	64284	65532	65295	66010	66933	67589	67551	68301	**71419**	66990.4

Table 7 – Computational times (seconds) for multiple intervals with variable length.

Instance	Amount of intervals (v)									Average
	10	9	8	7	6	5	4	3	2	
100a	2.0	1.6	1.4	0.3	0.0	0.0	0.0	0.1	0.1	0.6
200a	8.4	7.0	6.5	0.7	1.4	1.6	1.5	1.0	1.1	3.2
300a	21.0	17.7	13.5	10.8	7.3	5.3	7.6	8.9	12.2	11.6
400a	32.3	27.2	22.0	13.7	9.1	12.2	8.9	8.4	98.5	25.8
500a	76.3	53.3	46.2	36.4	24.0	33.9	27.6	358.0	1340.6	221.8
600a	73.2	59.4	68.5	58.5	51.1	110.0	205.9	495.3	21043.5	2462.8
700a	84.5	75.3	59.1	51.0	44.0	40.9	264.7	71.7	1069.3	195.6
800a	151.8	117.6	97.2	88.7	195.0	88.0	174.5	5548.7	10840.0	1922.4
900a	133.9	111.6	108.9	96.2	104.9	84.5	278.6	1485.6	13762.2	1796.3
1000a	156.8	132.2	115.2	108.4	97.1	106.0	91.1	1283.5	4404.4	721.6

heuristic behavior. The computational time spent by the best partitioning is much shorter than the time spent by full optimization of instances. The computational results are relatively very close, which validates the algorithm proposed in this paper. The result obtained for instance 800a is remarkable. The partitioning method achieved a better result than the primal bound obtained by optimization of the mathematical formulation. The limit of 50000.0 seconds was achieved by CPLEX without having completed the optimization of the instance. According to [23], the dual bound for this instance was 38729.8. It is likely that the result obtained by partitioning is not the global optimum, but the method proposed in this article could outperform the CPLEX.

6 CONCLUDING REMARKS

The proposed hybrid methods use only the CPLEX optimizer combined with partitions (intervals) of the planning horizon. When a partition is executed, the tasks activated until the end of

Table 8 – Comparison between the best partitioning scheme and the best primal bound known so far.

	Result		Time (sec)	
Inst.	Partitioning	CPLEX	Partitioning	CPLEX
100a	304	304	0.0	0.1
200a	635	**636**	0.5	6.5
300a	2042	**2073**	5.7	196.3
400a	6641	**7271**	0.9	467.8
500a	14225	**14336**	342.5	6688.7
600a	10090	**10157**	172.4	50000.0
700a	31329	**31820**	164.7	50000.0
800a	**38511**	38351	6616.0	50000.0
900a	38684	**38733**	4072.4	50000.0
1000a	70519	**71864**	488.1	50000.0

the interval are fixed and the following interval runs. This algorithm is executed until the end of the planning horizon.

To perform the divisions of the horizon, three strategies have been proposed. The first one works with fixed-size partitions, i.e., all partitions (except the last) must have the same number of time units. The second strategy analyzes the number of tasks that can start their activation in each unit time. According to the number of partition required by the user, an average amount of tasks in each interval is calculated. The consecutive time units are grouped so as to contain approximately this amount of tasks regardless of the length that will have this partition.

The third approach is an extension of the second one. An initial partitioning is defined and implemented as the previous strategy. After the initial execution, the first interval is shortened and proceeds to a new execution. Then, the interval is lengthened compared to the original size and another run happens. Next, the algorithm moves to the second interval performing the same two procedures until the penultimate interval is considered. It is considered therefore repeated executions of the second strategy, with minor differences in length of the intervals.

The results showed that the second strategy produces better average results than the first one and is executed in less time. In fact, the CPLEX performance for each interval depends much more on the amount of tasks (and consequently of variables) to be analyzed than units of time.

The average results of the third strategy are still a bit better than the second, but the computational times grow significantly. Making a comparison of the cost-benefit ratio of the second and third strategies, the second one was chosen as the most advantageous strategy for partitioning the planning horizon.

The comparison of the best partitioning strategy with the full planning horizon utilization showed that the partitioning is a valid strategy and presents very satisfactory results with heuristic behavior.

ACKNOWLEDGMENTS

This work was partially supported by FAPERJ (grant E-26/112.411/2012).

REFERENCES

[1] ALCARAZ J & MAROTO C. 2001. A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, **102**(1): 83–109.

[2] ALVAREZ-VALDES R, CRESPO E, TAMARIT JM & VILLA F. 2006. A scatter search algorithm for project scheduling under partially renewable resources. *Journal of Heuristics*, **12**(1): 95–113.

[3] BOCTOR FF. 1990. Some efficient multi-heuristic procedures for resource constrained project scheduling. *European Journal of Operational Research*, **49**(1): 3–13.

[4] BOULEIMEN K & LECOCQ H. 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, **149**(3): 268–281.

[5] DAMAK N ET AL. 2009. Differential evolution for solving multi-mode resource-constrained project scheduling problems. *Computers & Operations Research*, **36**(9): 2653–2659.

[6] DEMEULEMEESTER E & HERROELEN W. 1992. A branch-and-bound procedure for multiple resource-constrained project scheduling problem. *Management Science*, **38**(12): 1803–1818.

[7] FEO T & RESENDE M. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, **6**: 109–133.

[8] GOLFETO RR, MORETTI AC & SALLES NETO LL. 2009. A genetic symbiotic algorithm applied to the one-dimensional cutting stock problem. *Pesquisa Operacional*, **29**(2): 365–382.

[9] GONÇALVES JF, RESENDE MGC & MENDES JJM. 2011. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics*, **17**(5): 467–486.

[10] HEBERT JE & DECKRO RF. 2011. Combining contemporary and traditional project management tools to resolve a project scheduling problem. *Computers & Operations Research*, **38**(1): 21–32.

[11] HOMBERGER J. 2007. A multi-agent system for the decentralized resource-constrained multi-project scheduling problem. *Int. Trans. in Oper. Res.*, **14**(3): 565–589.

[12] KOLISCH R. 1996. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, **14**(1): 179–192.

[13] LIN S-W, LEE Z-J, YING K-C & LEE C-Y. 2009. Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications*, **36**(2): 1505–1512.

[14] MINGOZI A ET AL. 1998. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, **44**(5): 714–729.

[15] MLADENOVIĆ N, BRIMBERG J, HANSEN P & PÉREZ JAM. 2007. The p-median problem: A survey of metaheuristic approaches. *European J Operational Research*, **179**(3): 927–939.

[16] NET – INSTITUCIONAL. 2013. `http://www.netcombo.com.br/institucional`

[17] NONOBE K & IBARAKI T. 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: RIBEIRO C & HANSEN P (editors), *Essays and Surveys in Metaheuristics*, pp. 557–588.

[18] POORZAHEDY H & ROUHANI OM. 2007. Hybrid meta-heuristic algorithms for solving network design problem. *European J, of Operational Research*, **182**(2): 578–596.

[19] ROSING KE & HODGSON MJ. 2002. Heuristic concentration for the p-median: an example demonstrating how and why it works. *Computers & Operations Research*, **29**(10): 1317–1330.

[20] SABINO JA, LEAL JE, STÜTZLE T & BIRATTARI M. 2010. A multi-objective ant colony optimization method applied to switch engine scheduling in railroad yards. *Pesquisa Operacional* **30**(2): 486–514.

[21] SILVA ARV & OCHI LS. 2007a. Effective grasp for the dynamic resource-constrained task scheduling problem. In: *Proc. of International Network Optimization Conference (INOC)*, Spa (Belgium).

[22] SILVA ARV & OCHI LS. 2007b. A hybrid evolutionary algorithm for the dynamic resource constrained task scheduling problem. In: *Proc. of the International Workshop on Nature Inspired Distributed Computing (NIDISC'07)*, LongBeach (EUA).

[23] SILVA ARV & OCHI LS. 2010. Hybrid heuristics for dynamic resource-constrained project scheduling problem. In: *Proc. of the 7th International Workshop on Hybrid Metaheuristics*, Viena (Austria).

[24] (2011) SILVA RC, CANTÃO LAP & YAMAKAMI A. Application of an iterative method and an evolutionary algorithm in fuzzy optimization. *Pesquisa Operacional* **32**(2): 315–329.

[25] VALLS V, BALLESTÍN F & QUINTANILLA S. 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, **185**(2): 495–508.

[26] YANASSE HH. 2013. A review of three decades of research on some combinatorial optimization problems. *Pesquisa Operacional*, **33**(1): 11–36.