A Parallel Branch and Bound algorithm for the Resource Levelling Problem with minimal lags

J. L. Ponz-Tienda, A. Salcedo-Bernal

Universidad de Los Andes, Carrera 1 Este No. 19A-40, Bogotá, Colombia

&

Eugenio Pellicer

Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain

Abstract: *The efficient use of resources is a key factor to minimize the cost while meeting time deadlines and quality requirements; this is especially important in construction projects where field operations make fluctuations of resources unproductive and costly. Resource Leveling Problems (RLP) aim to sequence the construction activities that maximize the resource consumption efficiency over time, minimizing the variability. Exact algorithms for the RLP have been proposed throughout the years to offer optimal solutions; however, these problems require a vast computational capability ("combinatorial explosion") that makes them unpractical. Therefore, alternative heuristic and metaheuristic algorithms have been suggested in the literature to find local optimal solutions, using different libraries to benchmark optimal values; for example, the Project Scheduling Problem LIBrary (PSPLIB) for minimal lags is still open to be solved to optimality for RLP. To partially fill this gap, the authors propose a Parallel Branch and Bound algorithm for the RLP with minimal lags to solve the RLP with an acceptable computational effort. This way, this research contributes to the body of knowledge of construction project scheduling providing the optimums of 50 problems for the RLP with minimal lags for the first time, allowing future contributors to benchmark their heuristics methods against exact results by obtaining the distance of their solution to the optimal values. Furthermore, for practitioners, the time required to solve this kind of problems is reasonable and practical, considering that unbalanced resources can risk the goals of the construction project.*

Keywords: *Branch and Bound, Benchmarking, Construction Project, Parallel Computing, Resource Levelling Problem, Virtual Computing.*

1 INTRODUCTION

In any construction project, the tangible resources — mainly materials, equipment and labor needed to implement the construction schedule— are generally constrained or limited (Hinze, 2012) (Benjaoran, et al., 2015). Even though the logical sequence of the activities shapes the initial schedule, resource allocation and levelling outlines the final timetable, resolving conflicts as well as balancing the workload throughout the construction project (Anagnostopoulos & Koulinas, 2010) (Hinze, 2012). Therefore, the goal of minimizing the cost, while fulfilling the total project duration (makespan) and achieving the approved performance, demands an efficient use of construction resources (Georgy, 2008) (Koulinas & Anagnostopoulos, 2013) (Tang, et al., 2014); by accomplishing this goal, the construction company remains competitive too (Hariga & El-Sayegh, 2010). Even though, resource scheduling problems can be considered recurrent in project management at large, they are especially important in construction (Doulabi, et al., 2010) (Jun & El-Rayes, 2011) where field operations make fluctuations of resources (peaks) very inefficient and costly on a short-term basis: hiring leads to low-quality workers with no learning curve, whereas heavy equipment cannot be rented or only at a very high cost.

The project management literature classifies resource project scheduling problems in two groups (Anagnostopoulos & Koulinas, 2010) (Hinze, 2012) (Damci, et al., 2013a) (Benjaoran, et al., 2015): a) the Resource Constrained Project Scheduling Problem (RCPSP henceforth); and b) the Resource Levelling Problem (RLP hereafter). On the one hand, the RCPSP aims to minimize the makespan considering the precedence relationships as constraints with a limited availability of resources. On the other hand, the RLP aims to offer the sequence that maximizes the resource consumption efficiency over time, minimizing the variability, with an unlimited availability of resources and a prescribed makespan. Taking into consideration both resource scheduling problems (the RCPSP and the RLP), the construction schedule can be more reliable to minimize resource fluctuations and fulfil the goals of the construction project (Damci, et al., 2013a) (Faghihi, et al., 2016).

With the aim to offer optimal solutions for the resource project scheduling problems, exact algorithms based upon enumeration, integer programming or mixed integer programing have been proposed by researchers along the literature, but this kind of NP-Hard problems has a phenomenon of "combinatorial explosion" (Rieck & Zimmermann, 2015) (Neumann, et al., 2003). In other words, a rapid non-polynomial acceleration increase in the number of possible solutions as a function of the number of activities and their total slack, especially for large problems (Ponz-Tienda, et al., 2013); this phenomenon is particularly significant in construction projects (Anagnostopoulos & Koulinas, 2010).

Although these algorithms produce the absolute optimum to a given problem, they are not functional from a practical point of view, as they require a vast computational capability. To cope with this issue, alternative heuristic and metaheuristic algorithms have been proposed in the literature to find local optimal solutions with an acceptable computational effort. To prove the goodness of these heuristic algorithms, different libraries have been developed to test and benchmark heuristic solutions with the optimal solutions, especially for the RCPSP. However, in the current literature, only the library with minimal and maximal time lags up to 30 jobs for the RLP has been solved to optimality (Rieck, et al., 2012). However, the PSPLIB (Project Scheduling Problem LIBrary) for minimal lags (Kolisch & Sprecher, 1996) a problem without temporal windows more suited for construction projects is still open to be solved to optimality for the RLP (Ponz-Tienda, et al., 2013).

Therefore, to partially fill this gap, the authors propose a Parallel *Branch and Bound* algorithm for the RLP with minimal lags. This algorithm puts forward a systematic and sequential tree search that does not process unnecessary branches. Parallel computing increases the computational capabilities taking advantage of the easily accessible current multiple-processor technology; the available computer infrastructure allows programs to be run in many processors at the same time (Adeli, et al., 1993). As stated by Adeli (2000) (p.7), the trend in parallel processing and distributed computing "[…] *should be more toward solution of large-scale and complicated real-life engineering problems, the kind of problems that cannot be solved readily by traditional uniprocessor computers*". This way, this research contributes to the body of knowledge of construction project scheduling in three facets: (a) proposing a parallel exact procedure to solve non-regular problems as the RLP with an acceptable computational effort; (b) providing a benchmarking set of solutions (50 problems taken from the PSPLIB) to test the goodness of the heuristic algorithms for the RLP; and (c) solving to optimality in a reasonable computational time a realistic building project, proving the possibility to be implemented in commercial and professional applications to help practitioners in the scheduling of real and complex construction projects.

To present this proposal appropriately, the following section provides the problem description of the RLP. Next Section exposes the state-of-knowledge regarding the RLP, with particular emphasis on construction projects. Then, Section 3 details the proposed Parallel *Branch and Bound* algorithm for the RLP with minimal precedence relationships. In the following Section, the implementation of the Parallel *Branch and Bound* algorithm, as well as the results of the experimentation, are explained. Section 5 compares and discusses the results. An example of implementation to a real construction project (with 71 activities) follows to allow the reader understand its application. Finally, conclusions, limitations and future research lines are drawn.

2. PROBLEM DESCRIPTION

For the remainder of this paper, construction projects are specified by activity-on-node networks $G = (V, A)$, where V is the set of vertices and A is the set of arcs. Vertex set $V = \{0, 1, \cdots, n, n+1\}$ consists of n activities (Eq 1) that have to be carried out without interruption, and two fictitious activities, j_0 and j_{n+1}, that represent the beginning and the makespan (completion time of the project), respectively. The set of arcs consists of pair of elements $A = \{a(i,j) | i < j, i, j \in [0, n+1]\}$ that represent the precedence relationships between activities. Additionally, each activity must be executed in d_j time units and without pre-emption. The literature background of this problem is examined in the next section. This way, the general formulation of the RLP considers the following elements:

1. The set N of activities (being n the total number of activities):
$$N = \{j_1, \cdots, j_n\} \quad (1)$$
2. The set D of durations (being n the total number of activities):
$$D = \{d_1, \cdots, d_n\} \quad (2)$$
3. The set T of periods of time in which these activities have to be distributed (being t_p the deadline of the project, from now on denoted $\bar{T}$):
$$T = \{t_1, \cdots, t_p\} | t_p = \bar{T} \quad (3)$$
4. The set R of resources (being k the total number of resources):
$$R = \{r_1, \cdots, r_k\} \quad (4)$$

5. The set RC of resources requirements for each activity (being k the total number of resources and n the total number of activities):
$$RC = \{\{rc_{11}, \cdots, rc_{k1}\}, \cdots, \{rc_{1n}, \cdots, rc_{kn}\}\} \quad (5)$$
6. The set C of cost associated to each resource (being k the total number of resources):
$$C = \{c_1, \cdots, c_k\} \quad (6)$$
7. The set SS of scheduled starting times of each activity along the elements of the set T, in such way that:
$$SS = \{ss_1, \cdots, ss_j, \cdots ss_n\} | es_j \leq ss_j \leq ls_j \quad (7)$$
Being es_j and ls_j the early and latest starting time of the activity j.
8. The set SH of possible shifts of each one of the activities over the early start (es_j) between zero and its total float (tf_j):
$$SH = \{sh_1, \cdots, sh_n\} | sh_j \in \{0, \cdots, tf_j\}, ss_j = es_j + sh_j \quad (8)$$
9. The function $r_i(S,t) | 1 \leq i \leq k$, is defined as the consumption (u_{it}) of the resource r_i in the period of time t, belonging to the set T, in such way that the consumption of the resource r_i throughout the project for a feasible schedule $S \in SS$ in a period t, is given by:
$$u_{it} = r_i(S,t) \quad (9)$$
10. The set AV of availabilities of the resources:
$$AV = \{av_{it} | 1 \leq i \leq k, 1 \leq t \leq \bar{T}\} \quad (10)$$
11. The set SA of schedulable activities with total float $(tf_j > 0)$ strictly greater to zero (being m the total number of schedulable activities.):
$$SA = \{j_1, \cdots, j_m\} \quad (11)$$

Once the elements that compose the problem are set, a general formulation for the objective function of the optimization problem could be a function $f[r_i(S,t)]$, which computes the consumption of the resource r_i (during the period of time t) for a feasible schedule $S \in SS$, for all the k resources of the project multiplied by its associated cost (c_i):

$$Minimize \sum_{i=1}^{k} c_i \cdot f[r_i(S,t)] \quad (12)$$

The function $f[r_i(S,t)]$ provides different ways of dealing with the RLP. The most usual criterion focuses on getting the resource consumption as levelled as possible by minimizing the sample variance or mean square error over an ideal reference. Consequently, a suitable formulation for Equation 12, given a set $AV = \{av_{it}\}$ of availabilities could be:

$$f[r_i(S,t)] = \frac{(u_{it} - av_{it})^2}{\bar{T}}; \; Min \sum_{i=1}^{k} c_i \cdot \frac{(u_{it} - av_{it})^2}{\bar{T}} \quad (13)$$

The previous formulation can be simplified taking into account the following:

$$c_i = 1, \forall i, 1 \leq i \leq k \quad (14)$$

$$av_{it} = \bar{u} = \frac{\sum_{i=1}^{T} u_{it}}{\bar{T}}, \forall 1 \leq i \leq k, 1 \leq t \leq \bar{T} \quad (15)$$

Then, applying Equation 15, the Equation 13 can be simplified as follows:

$$\sum_{i=1}^{k} (u_{it} - \bar{u})^2 = \sum_{i=1}^{k} {u_{it}}^2 - \bar{T} \cdot \bar{u} \geq 0 \Rightarrow \sum_{i=1}^{k} {u_{it}}^2 \geq 0$$
$$\Rightarrow Min \sum_{i=1}^{k} (u_{it} - \bar{u})^2 = Min \sum_{i=1}^{k} {u_{it}}^2 \quad (16)$$

An equivalent formulation for Equation 13, known as the *Method of Minimum Squares Optimization*, is written this way:

$$min \sum_{i=1}^{k} {u_{it}}^2, 1 \leq t \leq \bar{T} \quad (17)$$

The complete formulation for the mathematical model of the *Minimum Squares Optimization* method is comprised in Equation 18:

$$\begin{aligned} & min \sum_{i=1}^{k} {u_{it}}^2 \\ & Subject\ to: \\ & \quad SS_{n+1} \leq \bar{T} \\ & \quad SS_i + d_i + \gamma_{ij} \leq SS_j, \forall i\ successor\ of\ j \\ & \quad \gamma_{ij}\ being\ the\ lead/lag\ between\ i\ and\ j \end{aligned} \quad (18)$$

Different objective functions for $f[r_i(S,t)]$ have been proposed (Damci, et al., 2016) in order to measure the efficiency of the construction project sequence. The most common objective function is the *Method of Minimum Squares Optimization*, which minimizes the sum of squares of periodically resource usages providing an ideal uniform shape for the levelled construction resource consumption. Similarly, the *Absolute Deviations Method* intends to deliver an ideal uniform shape by minimizing the resource utilization from the targeted resource utilization level (Younis & Saad, 1996), whereas the *Overloaded Resource Problem* considers additional costs if a threshold for the resource use is surpassed (Rieck, et al., 2012). A different objective function, the *Resource Idle Days and Maximum Daily Resource Demand Method* (El-Rayes & Jun, 2009), provides a Gauss shape instead of a rectangular distribution, in which the purpose is to eliminate the resource's idle periods. Florez, Castro-Lacouture, & Medaglia (2012) propose the *Maximizing Labor Stability*, which aims to increase the extent of use of workers and job continuity by two alternatives: the first minimizes the maximal fluctuation of workers, and the second the sum of the fluctuations.

3. LITERATURE REVIEW

The *Method of Minimum Squares Optimization*, expressed in Equation 17, was introduced by Burgess and Killebrew (1962) using a heuristic algorithm in which the local optimal (near-optimality or approximation to the optimal) was determined by the set of scheduled starting times (SS) for each period project along the elements of the set *T* for a prescribed makespan. The Burgess and Killebrew proposal is a one-pass improvement algorithm with a parallel backward outline and latest finishing time, as the priority rule, and maximum total float, as the secondary one. This scheduling outline offers poor improvements over the initial scheduling, but most important is that it usually offers infeasible solutions because it does not preserve the precedence restrictions in the original formulation.

To avoid previous limitations, Harris (1978) proposed the *Method of Minimum Moment* (MOM), a new multi-pass heuristic algorithm with floats recovery that preserve the precedence restrictions with better results than the Burguess & Killebrew proposal. Later, Harris (1990) improved its own proposal with the *Packing Method* (PACK), which recognizes network interactions with a more in-depth analysis. Hiyassat (2000; 2001) presented a modification of the MOM with a different criterion for selecting the activity to be shifted, based on the amount of the activity's resources rate and the value of its free float. More recently, Christodoulou et al. (2009) has put forward the entropy-maximization, using the maximality and sub-additivity properties of the entropy function.

As alternative to heuristic procedures, metaheuristic algorithms are higher-level procedures designed to find sufficiently good solutions to an optimization problem with limited computation capacity and grounded in physical, biological and animal behavior. Several specific examples about metaheuristics applied to RLP and RCPSP in construction projects can be found in the literature, such as Grasp (Anagnostopoulos & Koulinas, 2011), genetic algorithms (Hegazy, 1999) (Leu, et al., 2000) (Gaitanidis, et al., 2016), scatter search and Path Relinking (Ranjbar, 2013), simulated annealing (Son & Skibniewski, 1999) (Anagnostopoulos & Koulinas, 2010), simulation algorithm (Lim, et al., 2014), or tabu search (Koulinas & Anagnostopoulos, 2013). In other line, Adeli & Karim (1997; 2001) and Adeli & Wu (1998) proposed the application of neural network and Adeli & Karim (2001) a model based on neurocomputing and object technologies to construction projects.

Other approaches related to construction projects deal with RLP in Line-of-Balance Scheduling (Damci, et al., 2013a; 2013b), linear projects (Tang, et al., 2014) (Georgy, 2008), highway projects (Arditi & Bentotage, 1996), considering uncertainty in activity durations (Li & Demeulemeester, 2014), allowing activity splitting (Hariga & El-Sayegh, 2010) (Alsayegh & Hariga, 2012) (Hossein Hashemi Doulabi, et al., 2010) (Son & Mattila, 2004) or considering generalized precedence relationships (Benjaoran, et al., 2015). Construction-related problems derived from multimode RLP were studied by Menesi & Hegazy (2014), whereas Heon Jun & El-Rayes (2011) analyzed those related to multiobjective optimization.

There are several exact algorithms available for the solution of the RLP with minimum and/or minimum-maximum time lags, which may be separated into implicit enumeration outlines and integer and mixed-integer programming models. On the one hand, as a first contribution on exact implicit enumeration outlines methods, Petrovic (1969) introduced a dynamic programming for the RLP with precedence constraints. On the other hand, Ahuja (1976) proposed a method that enumerates all combinations of construction activity start times for networks with precedence constraints to minimize the squared changes in the resource utilization for minimum time lags. Later, Bandelloni, Tucci, & Rinaldi (1994) applied non-serial dynamic programming and interaction graph theory to find a minimum for the squared deviation from the average resource utilization. Son & Mattila (2004) proposed a linear program binary variable model to level construction resources that permits selected activities to stop and restart, resulting in an improvement of the leveling solution.

Additionally, models proposing *Branch and Bound* (B&B from now on) procedures and tree-based enumeration were presented by Nübel (2001). This proposal was adapted by Neumann, Schwindt & Zimmermann (2003) for the RLP to find a nearly optimal solution computing a lower bound for the objective function value of each partial tree in the enumeration. Gather et al. (2011) considered a tree-based enumeration outline where different techniques for avoiding redundancies were employed. None of these three contributions were specific for construction projects.

Alternatively, for the integer and mixed-integer programming models, based on the Pritsker et al. (1969) formulation, Easa (1989) developed a mixed binary-integer linear optimization model that minimizes the absolute deviations between the construction resource requirements and a desirable resource level (uniform or non-uniform). Next, Rieck et al. (2012) proposed a new mixed-integer linear model for RLP with domain-reducing pre-processing techniques; they solved, for the first time to optimality, the Kolisch et al. (1999) test set instances considering a deadline equal to the unconstrained makespan. Gather et al. (2011) presented a new tree-based enumeration outline, based on an extended bridge that enumerates all quasi-stable schedules without redundancy. Finally, Ponz-Tienda et al. (2013) proposed two different binary optimization models: the first uses binary decision variables x_{jst} that establish the period in which the construction activities are finished; and, in the second model, the decision variables x_{jst} establish the period in which the activities are executed.

However, the RLP is NP-hard in the strong sense and difficult to solve to optimality (Neumann, et al., 2003) (Rieck, et al., 2012). Additionally, the RLP is an especial case of the Project Scheduling Problems with non-regular objective functions (Neumann & Zimmermann, 1999; 2000) (Rieck & Zimmermann, 2015). This kind of problems cannot be solved by pruning the exploration tree with the traditional B&B procedures. *Branch and Bound* algorithms check the branches of exploration tree against the bounds on the optimal solution, and branches are discarded (cut) if they cannot produce a better solution than the best bound found so far by the algorithm. Consequently, with non-regular objective functions, a better solution can be found in the direction of a branching node with a worst solution, than the best bound found so far and, therefore, this solution cannot be discarded. This fact implies that the RLP could be even more difficult to solve than classical RCPSP, even when the initial universe of possible solutions to the problem is less than in the resource-constrained case. To deal with this problem, relaxations to the problem and heuristic procedures have been proposed along the literature to find near optimal solutions to the RLP.

As relaxations to the RLP, Drexl & Kimms (2001) developed two methods for lower bound computations: the first is based on a Lagrangian relaxation, whereas the second is based on a column generation procedure, where variables represent schedules. Coughlan et al. (2010) proposed a *Branch-and-Price* algorithm by column generation embedded into a B&B outline, building a pricing problem for each shift of the activities. The same authors (Coughlan, et al., 2013) improved their previous proposal with a linear programming relaxation based on variables that represent schedules via a column generation and *Branch-and-Price* algorithm. Later, Yeniocak (2013) proposed a B&B algorithm with a lower bound calculation strategy and a dual calculation to obtain lower bound values using the *Resource Idle Days* and *Maximum Daily Resource Demand* (RID-MRD) metric for problems up to 20 activities. Such relaxations do not assure a global optimal solution for the RLP; therefore, an alternative possibility to obtain the global optimum is to divide the original problem into simpler sub-problems and solve them using parallel computation.

Parallel computing has been previously proposed and used to "upgrade" algorithms in civil engineering and make them perform faster. The first approach to this topic in civil engineering was presented by Adeli and Vishnubhotla (Adeli & Vishnubhotla, 1987). Later, Adeli and Kamal (1989) introduced a parallel algorithm for structural analysis and its performance results. Both approaches made use of the parallel capabilities of available "supercomputers", which are a very limited resource. Nevertheless, today's technology makes possible to connect multiple personal computers (which are cheaper every day) with little effort; therefore, new approaches show up. These approaches take advantage of the abundance of personal computers in order to propose algorithms that can be run in multiple machines at the same time (Adeli & Kumar, 1999).

Although parallel computation has been widely studied for structural analysis (Adeli, et al., 1993), integer programming (Wah, et al., 1985) and parallel B&B (McKeown, et al., 1991) (Clausen & Perregaard, 1999) (Crainic, et al., 2006) (Ismail, et al., 2014), proposals on parallel computing in scheduling are scarce and based on the parallelization of the B&B procedure (Perregaard & Clausen, 1998) (Chakroun & Melab, 2015), but not on the subdivision of the graph.

Moreover, the RLP with minimal lags is harder to solve than the minimal-maximal one because, in the problem with minimal lags, the activities have more freedom in their shifts and consequently a greater universe of feasible solutions. Finally, in the reviewed literature, only the library with minimal and maximal time lags (Kolisch & Sprecher, 1996) up to 30 jobs has been solved to optimality, considering a deadline equal to the unconstrained makespan (Rieck, et al., 2012); nonetheless, the PSPLIB (Kolisch & Sprecher, 1996) library for minimal lags is still open to be solved to optimality (Ponz-Tienda, et al., 2013). Consequently, to partially fill this gap and make a contribution to the body of knowledge, the authors propose in the next Section a Parallel *Branch and Bound* algorithm for the Resource Levelling Problem with minimal lags. Furthermore, this algorithm is tested with 50 problems of the PSPLIB providing a benchmarking set of solutions, as described in Section 5.

4. PROPOSED PARALLEL EXACT PROCEDURE

A complex problem is easier to deal with and more efficient to solve when divided into simpler sub-problems. This paradigm is especially efficient combined with parallel computing algorithms, converting a sequential problem into parallel processing sub-problems, using multiple and independent processors all of them running their own sub-problem at the same time and then merging its respective results.

Parallel *Branch and Bound* (B&B) algorithms present some anomalies (Lai & Sahni, 1984) in such way that a problem of *n2* threads can take more time than a problem with *n1* threads, even though *n2* < *n1*. This is because the pruning process depends on the current best bound found, and the parallelization causes the unnecessary processing of branches with worst solutions. However, for non-regular problems as the RLP, this is not an issue because all the solutions of the exploration tree must be analyzed, and therefore, the parallelization process is efficient.

The proposed parallel procedure is based on a cloud and multicore network computing to work simultaneously with various sub-problems of a given problem using a structure of parallel processing and distributed computing (Adeli & Kumar, 1995) (Adeli, 2000). This approach makes use of multiple execution units on the same processor (threads) and multiple processors with a sub-problems manager and communication over the Internet, as depicted in Fig 1.

4.1. Graph subdivision process

The process of breaking down a project graph into smaller sub-project graphs, simple enough to be solved in a reasonable computational effort, consists in a multi-branched recursive process. Then, the solutions of the sub-problems are combined to give the solution of the original problem. The subdivision is based on the assumption that all the activities can be scheduled on every position along its total float. From this assumption, it is possible to establish a one to one correspondence between possible configurations of n activities and integer sequences of n terms, where the i^{th}

term is between zero and the total float (tf_i) of the i^{th} activity.

Let s be a sequence of integer positive values (Eq 19). Each one of the elements of the sequence s represents a possible shift (not necessarily feasible) for activity between zero and its total float (tf) for an arbitrary problem with n activities:

$$s = \{s_1, \cdots, s_i, \cdots, s_n\} | s_1 \in \{0, \cdots, tf_1\}, \cdots, s_i \in \{0, \cdots, tf_i\}, \cdots, s_n \in \{0, \cdots, tf_n\} \quad (19)$$

Let be S the set of elements of s possible solutions (sequences) (Eq 20):

$$S = \{s | s \; represent \; a \; possible \; solution\} = \{[0, f_1], \cdots, [0, f_i], \cdots, [0, f_n]\} \quad (20)$$

The process of subdivision of the set starts selecting a i^{th} position. In this way, the set S can be divided in two new disjoint subsets from the floor function of its middle point of the i^{th} position, as stated in Equation 21:

$$\begin{gathered} S = S' \cup S'' \\ S = \left\{[0, f_1], \cdots, \left[0, \left\lfloor\frac{f_i}{2}\right\rfloor\right], \cdots, [0, f_n]\right\} \cup \left\{[0, f_1], \cdots, \left[\left\lfloor\frac{f_i}{2}\right\rfloor + 1, f_i\right], \cdots, [0, f_n]\right\} \\ |S'| \geq |S''| \end{gathered} \quad (21)$$

In addition, the new subsets are disjoint sets, in such way that if a possible sequence is in $S´$ cannot be found on $S´´$ and vice versa (Eq 22):

$$\left\{[0, f_1], \cdots, \left[0, \left\lfloor\frac{f_i}{2}\right\rfloor\right], \cdots, [0, f_n]\right\} \cap \left\{[0, f_1], \cdots, \left[\left\lfloor\frac{f_i}{2}\right\rfloor + 1, f_i\right], \cdots, [0, f_n]\right\} = \emptyset \quad (22)$$

The division process could also be applied recursively to intervals in $[f_1, f_2] | f_2 > 0, f_2 > f_1$ form, where the intervals gets divided by middle point, up to a previously established depth m, obtaining 2^m subsets, as displayed in Eq 23:

$$2^m \leq \prod_{i=1}^{n} (f_1 + 1) \quad (23)$$

Being $\prod_{i=1}^{n} (f_1 + 1)$ the universe of possible schedules for an instance (Ponz-Tienda, et al., 2013). The subdivision process could produce not feasible subsets, in such way that there are not any sequences in these subsets that meet the constraints and, consequently, their branches are discarded. The subdivision process of the set S on subsets of possible solutions is shown in Figure 2.

The proposed algorithm for the graph division, being $S = \{[l_1, u_1], \cdots, [l_n, u_n]\}$ a set of sequences of integer positive values ($S_{li} = l_i$ and $S_{ui} = u_i$), is shown in Pseudo-code 1, and the algorithm for compute the times and analyze the feasibility of a sequence is displayed in Pseudo-code 2.

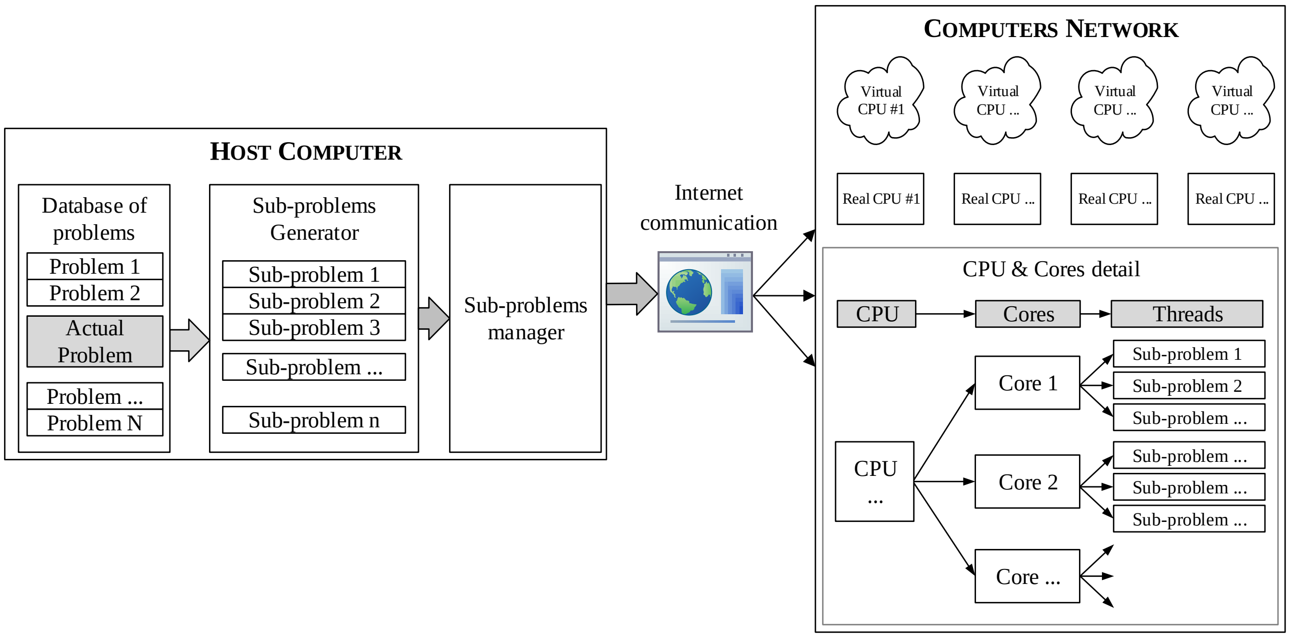


Figure 1 Parallel processing

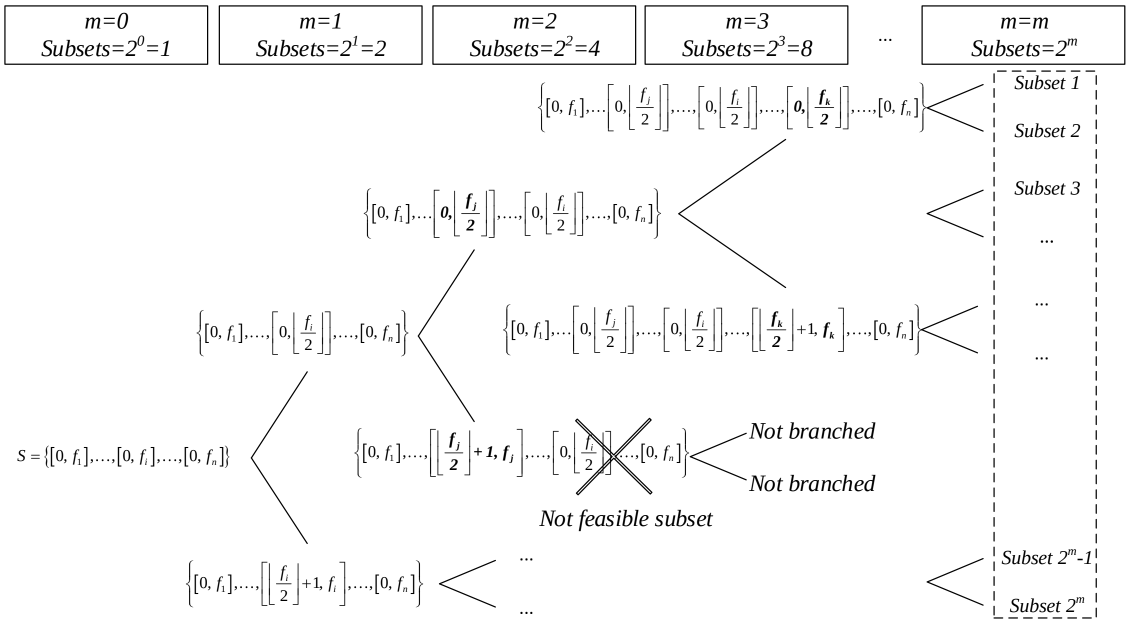


Figure 2 Subdivision process of the set S on subsets of possible solutions

Pseudo-code 1
Algorithm for graph division

```
GraphDivide (initialsequence:S, divisiondegree:m)
    Queueofsequences: Q
    Divisionindex: i = 0
    Lastdivisionindex: l = 0
    Q.enqueue(S)
    while (m > 1)  do
        'remark Search for a suitable index i to divide all sequences in Q
        i = 0
        for j = l + 1 to N
            if (min{ $T_{uj}$ - $T_{lj}$: T ∈ Q } > 0)  then
                i = j
                j = N + 1
            end if
        end for
        if (i > 0)  then
            count = |Q|
            'remark Divide all sequences in Q by the index i
            while (count > 0)  do
                T = Q.dequeue()
                U = clone(T)
                middle = ⌊($T_{ui}$ - $T_{li}$ )/2⌋
                $T_{ui}$ = middle
                $U_{li}$ = middle + 1
                if (Computetimes(T) = true)  then Q.enqueue(T)
                if (Computetimes(U) = true)  then Q.enqueue(U)
                count = count - 1
            end while
            l = i
        elseif (l = 0)  then break
        else l = 0
    end while
return Q
```

Pseudo-code 2
Computation times to analyze the feasibility of a sequence

```
function Computetimes (sequenceset:S)
    for i = 1 to N
        $SS_i$ = max{$SS_i$, $ES_i$ + $S_{li}$ }
        if ($SS_i$ > $ES_i$ + $S_{ui}$)  then return false
        for each j in successors(i)
            $SS_j$ = max{$SS_j$, $ES_i$ + $d_i$ )
        end for
    end for
return true
```

The proposed approach to subdivide the graph aims to produce similar sized subgraphs without building all the exploration trees that cannot be efficiently managed due to their great size. In this way, the goal is to find an index (activity) such that it is possible to divide all subgraphs by the same activity. However, although the number of possible solutions of each subset will be similar, the number of feasible solution could differ widely as exposed in the following section.

Once the division process is finished, the “sub-problems manager” (Fig. 1) begins the distribution over the computer network delivering to each computer as many sub-problems as cores (processors) available. The remaining sub-problems are kept on a distribution queue waiting to be processed. The manager makes a periodical checks (once every two seconds) to determine if is possible to deliver new sub-problems, because some of them have already finished or some computers have been disconnected: in the first case, delivering new sub-problems, and in the second, delivering the unprocessed sub-problems. Additionally, the sub-problems manager storages the optimal values obtained so far, merging its respective results. The process for the graph subdivision and sub-problems distribution is insignificant respect to the time required solving the problems; as pointed out by Adeli and Kamal (1992), this is a requirement for an efficient concurrent algorithm. As example, a problem of 10^{18} can take an overage of one second to divide and distribute it, and over one month to solve it with one hundred cores.

4.2. Example of graph subdivision process

For a better understanding of the subdivision process, Figure 3 displays a Gantt diagram where black squares represent critical activities, dark grey the non-critical ones, and on light grey the total floats.

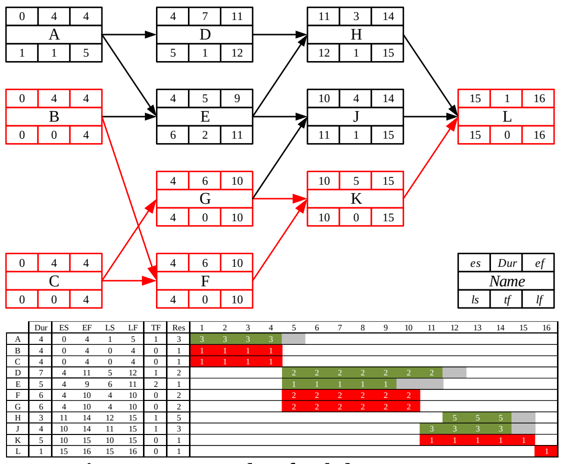


Figure 3 Example of subdivision process

The Example shown in Figure 3 could be divided for an arbitrary depth ($m = 3$), remarking in light gray the position in the subset in which the subset was branched (Fig 4).

Not necessarily all the subsets are feasible subsets, and not all the sequences of feasible subsets are feasible sequences (Fig. 4) Therefore, the branching process cut the not feasible branches with not feasible solutions, and not necessarily all the sequences of feasible subsets are feasible sequences. On the Example of Figure 3, the sequence {0,0,0,0,2,0,0,0,0,0,0} in #$S2$, given by shifting the fifth activity (E) two-steps is a possible sequence, but not a feasible sequence, due to the fact that this shift violates the precedence constraints with its follower (J).

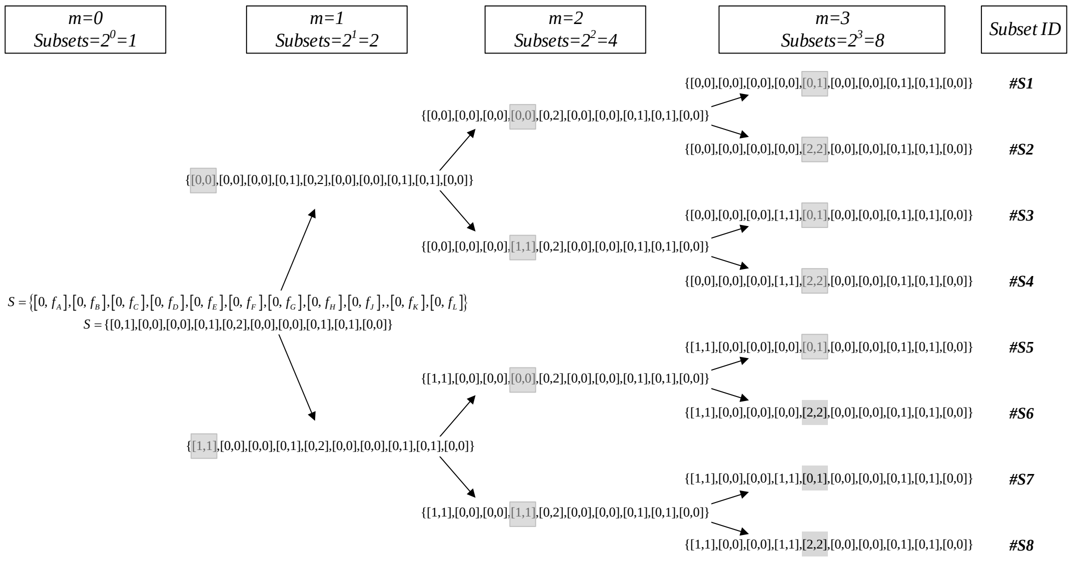


Figure 4 Process subdivision tree of the graph Example

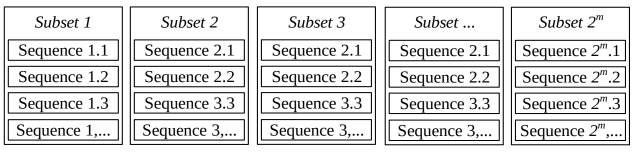


Figure 5 Composition of possible sequences for feasible subsets of the problem

4.3. The implicit enumeration outline

Once the problem is divided in feasible subsets, the process of *Implicit Enumeration* starts for each one of them.

The term *Implicit Enumeration* implies that not all the solutions of the enumeration outline are analyzed and that large numbers of not feasible solutions are excluded. Note that in Figure 6 each node represents an activity and its shift along the restricted total float in the subset.

The scanning sequence for the *Implicit Enumeration* of the subsets is a tree enumeration with outline *Depth First Search* in which the live node with deepest level in the search tree is chosen for exploration (Fig 7).

Table 1
Sequences for all subsets for the example of subdivision process

Sub Set #S1	*Sub Set #S2*	*Sub Set #S3*	*Sub Set #S4*	*Sub Set #S5*	*Sub Set #S6*	*Sub Set #S6*	*Sub Set #S6*
{0,0,0,0,0,0,0,0,0,0,0}	{0,0,0,0,2,0,0,0,0,0,0}	~~{0,0,0,1,0,0,0,0,0,0,0}~~	~~{0,0,0,1,2,0,0,0,0,0,0}~~	~~{1,0,0,0,0,0,0,0,0,0,0}~~	~~{1,0,0,0,2,0,0,0,0,0,0}~~	~~{1,0,0,1,0,0,0,0,0,0,0}~~	~~{1,0,0,1,2,0,0,0,0,0,0}~~
{0,0,0,0,0,0,0,0,1,0,0}	~~{0,0,0,0,2,0,0,0,1,0,0}~~	~~{0,0,0,1,0,0,0,0,1,0,0}~~	~~{0,0,0,1,2,0,0,0,1,0,0}~~	~~{1,0,0,0,0,0,0,0,1,0,0}~~	{1,0,0,0,2,0,0,0,1,0,0}	~~{1,0,0,1,0,0,0,0,1,0,0}~~	~~{1,0,0,1,2,0,0,0,1,0,0}~~
{0,0,0,0,0,0,0,1,0,0,0}	{0,0,0,0,2,0,0,1,0,0,0}	{0,0,0,1,0,0,0,1,0,0,0}	~~{0,0,0,1,2,0,0,1,0,0,0}~~	~~{1,0,0,0,0,0,0,1,0,0,0}~~	~~{1,0,0,0,2,0,0,1,0,0,0}~~	~~{1,0,0,1,0,0,0,1,0,0,0}~~	~~{1,0,0,1,2,0,0,1,0,0,0}~~
{0,0,0,0,0,0,0,1,1,0,0}	~~{0,0,0,0,2,0,0,1,1,0,0}~~	{0,0,0,1,0,0,0,1,1,0,0}	{0,0,0,1,2,0,0,1,1,0,0}	~~{1,0,0,0,0,0,0,1,1,0,0}~~	{1,0,0,0,2,0,0,1,1,0,0}	~~{1,0,0,1,0,0,0,1,1,0,0}~~	{1,0,0,1,2,0,0,1,1,0,0}
{0,0,0,0,1,0,0,0,0,0,0}		~~{0,0,0,1,1,0,0,0,0,0,0}~~		{1,0,0,0,1,0,0,0,0,0,0}		~~{1,0,0,1,1,0,0,0,0,0,0}~~	
{0,0,0,0,1,0,0,0,1,0,0}		~~{0,0,0,1,1,0,0,0,1,0,0}~~		{1,0,0,0,1,0,0,0,1,0,0}		~~{1,0,0,1,1,0,0,0,1,0,0}~~	
{0,0,0,0,1,0,0,1,0,0,0}		{0,0,0,1,1,0,0,1,0,0,0}		{1,0,0,0,1,0,0,1,0,0,0}		{1,0,0,1,1,0,0,1,0,0,0}	
{0,0,0,0,1,0,0,1,1,0,0}		{0,0,0,1,1,0,0,1,1,0,0}		{1,0,0,0,1,0,0,1,1,0,0}		{1,0,0,1,1,0,0,1,1,0,0}	

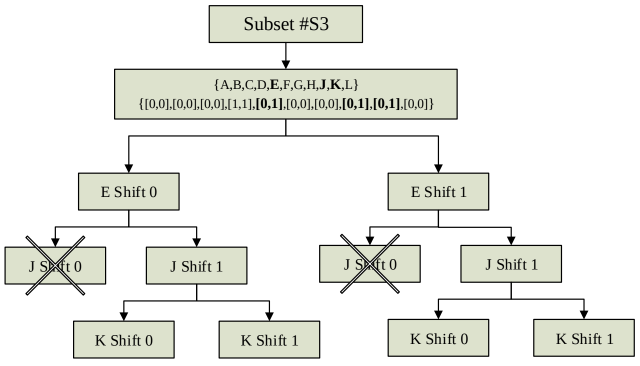


Figure 6 Tree representation of the subset #S3

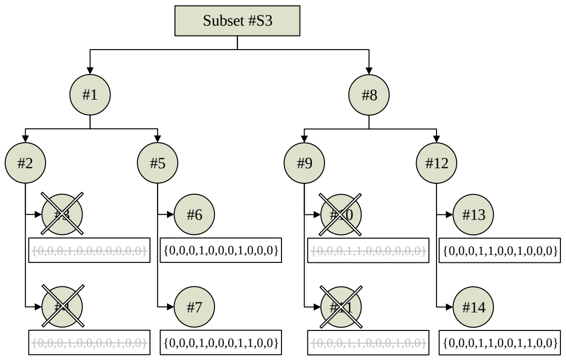


Figure 7 Depth First Search order enumeration outline for Sub Set #3 and obtained sequences

Pseudo-code 3 shows the algorithm for the *B&B* with *Depth First Search* outline. The algorithm is a recursive procedure, in which nodes are examined incrementally from a starting index (initial job for the first call) while it can be branched, calling himself from the next index. The *B&B* algorithm (non-parallel) with *Depth First Search* outline for the RLP has been implemented in an app (Fig 8) as an illustrative example of the search procedure that can be downloaded from https://goo.gl/hxW8c9.

Pseudo-code 3

Depth First Search outline Branch and Bound algorithm

```
ParallelBranch&Bound (jobindex:i, sequenceset:S)
'remark Tries with the possible shifts of ith  activity given by S constraints
    for k = S_li  to S_ui
        if (ES_i + k < SS_i)  then
            k = SS_i - ES_i
        else
            SS_i = ES_i + k
        end if
        'remark The ith  activity shifting could induce the violation
        'of another activity constrain in S
        isfeasible = Computetimes(S)
        if (isfeasible = true) then actualbound = Computeobjectivefunction()
        if (actualbound < lowerbound) then
            lowerbound = actualbound
            for z = 1 to N
                Elite_z = SS_k
            end for
        end if
        'remark recursive calling
        if (i < N) then
            call ParallelBranch&Bound (i + 1,S)
        else
            for z = i + 1 to N
                SS_z = 0
            end for
            'remark If a constraint gets violated, the next shifts will be
            'violated. So is no need to continue
            k = S_ui + 1
        end if
    end for
```

As it can be seen on Pseudo-code 3, there is not a previous process that excludes activities with zero total float or without resource consumption, because the subdivision process could reduce to zero the total float of some activities. Consequently, the real complexity could differ between the sequential and the parallel computation in a number that is less or equal to the number of feasible subsets. The application of Pseudo-codes 1 to 3 provides the feasible sequences presented in Table 1 for each one of the subsets of the Example.

In Table 2, the initial and levelled values for the objective function for each subset are shown. Note that, due to the fact that RLP is a non-regular problem, a lower bound cannot be established to cut the exploration tree. Therefore, all subsets must be analyzed in order to find the optimal value. In the example of graph subdivision process, the subsets #S2 and #S4 present the best initial value; however, the optimal value is found in subset #S3.

Table 2

Values for the objective function for each sub-set

	Initial value	*Best value*
#S1	956	926
#S2	920	920
#S3	922	916
#S4	920	920
#S5	988	968
#S6	962	962
#S7	944	938
#S8	932	932

5. IMPLEMENTATION AND RESULTS

The proposed *Parallel B&B* algorithm has been completely implemented in an app developed in C# language and tested with the PSPLIB library (Kolisch & Sprecher, 1996). The process of implementation of the *Parallel B&B* was gradual from one core to 490 cores (Fig 8). The first step (with one core) had the purpose of testing the *B&B* algorithm (Fig 9). Once the goodness of the provided solutions was verified, the parallel *B&B* was implemented in one computer with eight cores and gradually increased to 490 cores in several physical and virtual computers. The complete app with the *Parallel B&B* and the user manual can be downloaded from https://goo.gl/F0vKOL.

The *Information and Technology Services Directorate (DSIT)* and the *Department of Systems Engineering* of la *Universidad de Los Andes, Colombia*, provided 25 Windows 7© virtual machines of 10 cores and 2.4 GHz under the HP Cloud Management Software©. Furthermore, five Workstations HP Z620 12 DIMM (with a Chipset Intel® C602) and one additional CPU Xeon E5-2603v2 were used. Additionally, several *Linux* virtual machines were provided by *UnaCloud* under the *Opportunistic Cloud Computing Infrastructure as a Service* framework (Rosales, et al., 2011).

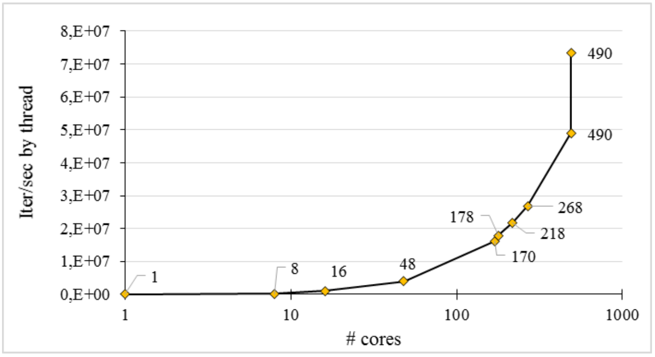
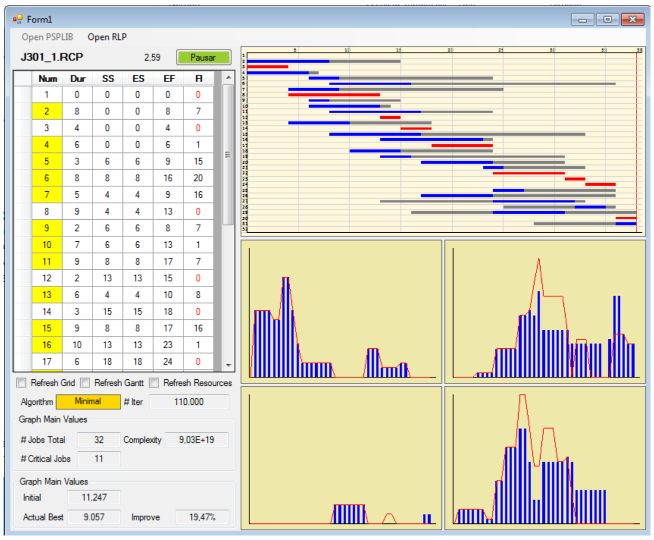


Figure 8 Improvement process considering the number of cores

Along with the implementation process, the algorithm was significantly improved, not only by increasing the number of cores, but also the efficiency in the iterations by thread, going from 5,000 to around 150,000 iterations per second and thread (Fig 10, left side), and 7.35E+07 total iterations per second (Fig 10, right side).



J301_1.RCP 2,59

Num	Dur	SS	ES	EF	FI
1	0	0	0	0	0
2	8	0	0	8	7
3	4	0	0	4	0
4	6	0	0	6	1
5	3	6	6	9	15
6	8	8	8	16	20
7	5	4	4	9	16
8	9	4	4	13	0
9	2	6	6	8	7
10	7	6	6	13	1
11	9	8	8	17	7
12	2	13	13	15	0
13	6	4	4	10	8
14	3	15	15	18	0
15	9	8	8	17	16
16	10	13	13	23	1
17	6	18	18	24	0

☐ Refresh Grid ☐ Refresh Gantt ☐ Refresh Resources

Algorithm: Minimal # Iter: 110.000

Graph Main Values
Jobs Total: 32 Complexity: 9,03E+19
Critical Jobs: 11

Graph Main Values
Initial: 11.247
Actual Best: 9.057 Improve: 19,47%

Figure 9 App for the Depth First Search B&B algorithm

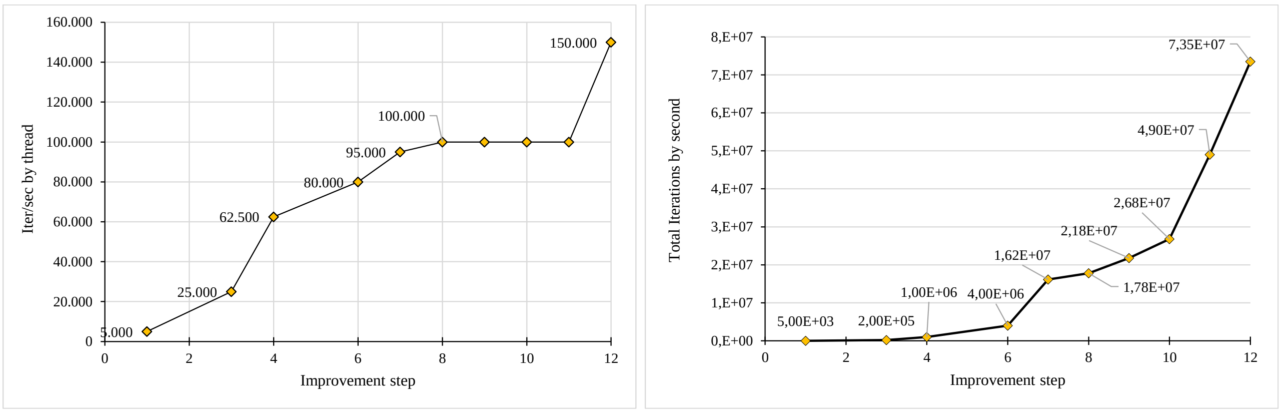


Figure 10 Improvement process in the number of iterations by second

For the graph subdivision, a non-fixed depth variable m (Eq. 23) is adopted in such way that the division process produces at least $2^m > 20{,}000$ feasible subgraphs. This ensures a balanced complexity for the analyzed problems with the available infrastructure, without disturbing the efficiency of the sub-problems manager negotiation process. The real complexity of different solved instances is shown in Figure 11. The x-axis represents the subgraph, and the y-axis the number of iterations needed to be solved; the red line displays the cumulated iterations. As shown in Figure 11, the complexity of the sub-problems can differ widely (up to 10 orders of magnitude).

By dividing the problem into a large number of sub-problems, the sub-problems manager minimizes idle times of the computer network and, additionally, reduces the difference of complexity between sub-problems, balancing the distribution of work among processors (Saleh & Adeli, 1994a; 1994b). If all sub-problems where equally sized, the perfect distribution would be to divide the problem in as many sub-problems as processors are available, mapping the sub-problems with the processors one to one (Adeli & Kamal, 2003). However, the sub-division process does not produce equally sized sub-problems, as can be seen in Figure 11, where there are sub-problems with a thousand times more complexity than others. Therefore, this distribution would produce idle time among the processors: if a sub-problem is considerably bigger, then its respective processors will spend more running time whereas the remaining processors are waiting. Thus, it is recommended to produce a big number of sub-problems to minimize this issue.

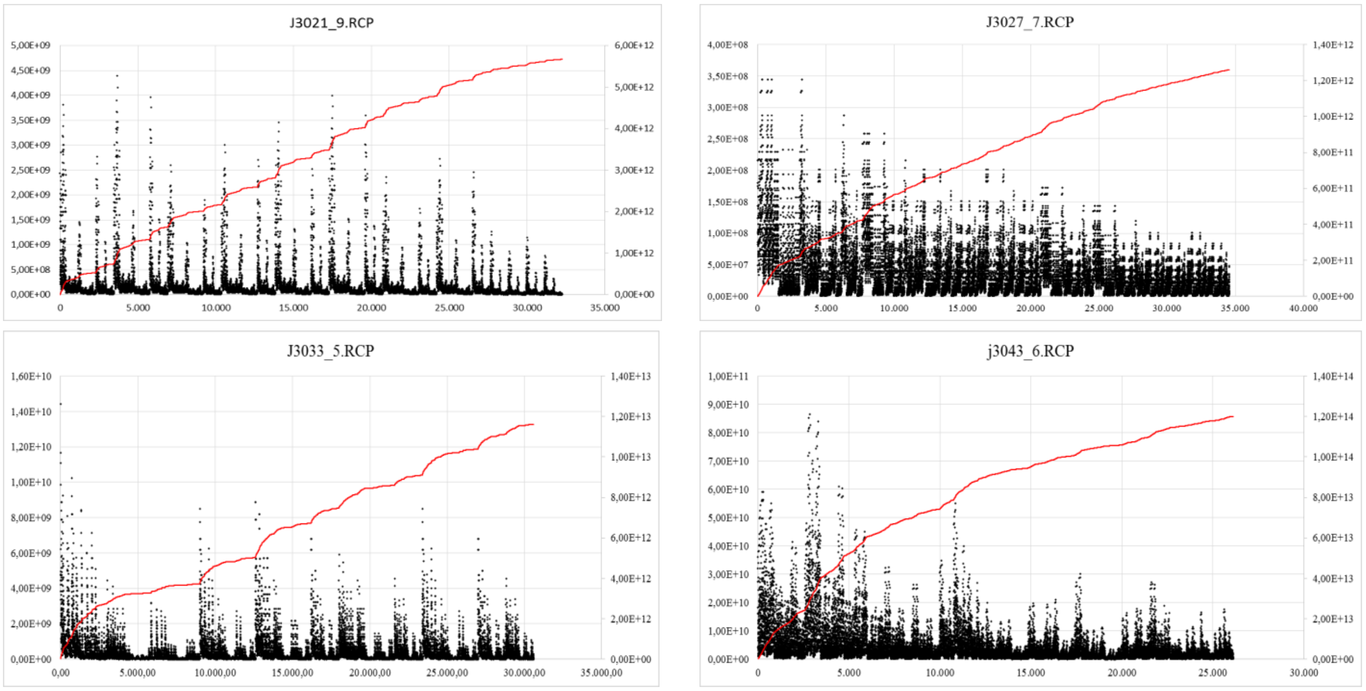


Figure 11 Real complexity of the subgraphs after graph subdivision

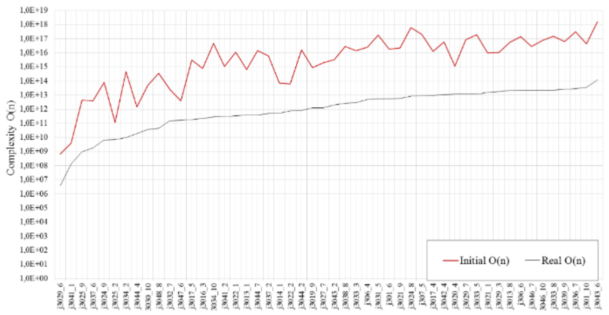


Figure 12 Comparison of the initial complexity over real

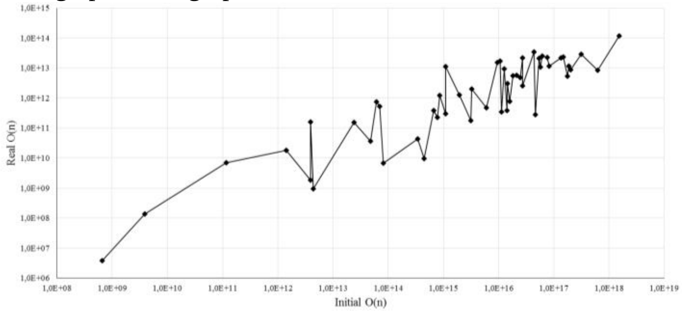


Figure 13 Correlation of the initial complexity over real

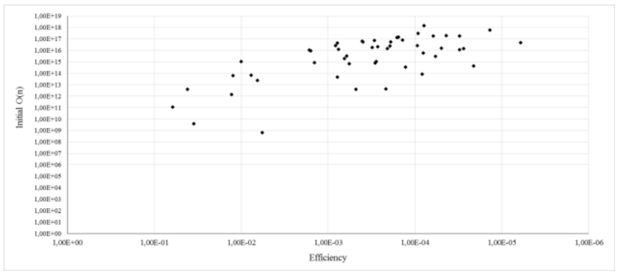


Figure 14 Correlation of the initial complexity over efficiency

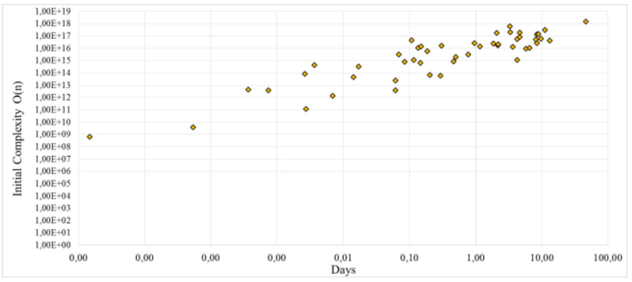


Figure 15 Comparison of the initial complexity over the processing time considering 3×10^7 iterations per second

For the experimentation phase, all problems with an initial complexity less than 10^{17} (42 instances) were selected, on top of seven arbitrary instances with a complexity between 10^{17} and 10^{18} and one instance with a complexity greater than 10^{18}. Table 3 exposes the main indicators for each one of the solved instances: (a) the initial complexity, as the product of the total floats of the activities $(O(n) = \prod(Totalfloat + 1))$; (b) the real number of iterations needed (feasible solutions); (c) the efficiency, as the relation between the real and initial complexity (smaller values correspond to greater efficiency); and (d) the initial and levelled values for the sum of the squares and the improvement as a percentage of the initial value. For a complete compatibility of the benchmarking test, the problems were solved considering the prescribed makespan as the obtained for the unconstrained problem $(\bar{T} = ls_n, equation\ 3)$ and the associated cost to each resource equal to one ($c_i = 1, equation\ 14$). The optimal values, the scheduled times for the activities, and the benchmarking results can be downloaded from https://goo.gl/GjCGEi.

Figures 12 and 13 show the correlation between the initial complexity versus the real number of iterations and the processing time, respectively. In Figure 12, the problems, in x-axis, are ordered by their real complexity and then compared with their respective initial complexity, without a clear relationship between those magnitudes. In Figure 13, the initial and real complexity of each solved problem are mapped showing that, in general, the real complexity is increasing with the initial complexity, but with a variability up to three levels of magnitude between two similar problems. This implies that the real complexity cannot be established by only considering the initial complexity. The processing time has been computed considering an average of 3×10^7 iterations per second.

Figures 14 and 15 display the correlation between the initial complexity over efficiency and the processing time, considering 3×10^7 iterations per second, respectively. Figure 14 shows the relationship between the efficiency and the initial complexity, with a weak correlation between the two metrics, indicating that bigger problems are more efficiently solved, but with great variability up to two levels of magnitude between similar problems. Figure 15 compares the time required by the instances to be solved to optimality versus its initial complexity, suggesting that problems up to 3×10^{17} can be solved in a reasonable computational time.

Table 3
Parallel B&B experimentation results

# of problem	*Problem*	*Complexity O(n)*		*Efficiency*	*Sum of squares*		*Improvement*
		Initial	Real		Initial	Levelled	
1	j3029_6	6.6E+08	3.8E+06	0.573%	110,265	107,109	2.862%
2	j3041_1	3.9E+09	1.4E+08	3.525%	45,832	41,878	8.627%
3	j3025_2	1.2E+11	7.1E+09	6.167%	39,398	37,178	5.635%
4	j3044_4	1.4E+12	1.8E+10	1.293%	51,951	49,379	4.951%
5	j3037_6	3.9E+12	1.9E+09	0.048%	22,127	19,399	12.329%
6	j3047_6	3.9E+12	1.6E+11	4.149%	76,849	71,273	7.256%
7	j3025_9	4.4E+12	9.5E+08	0.022%	42,302	40,714	3.754%
8	j3032_7	2.4E+13	1.6E+11	0.652%	75,573	66,411	12.123%
9	j3030_10	4.7E+13	3.7E+10	0.078%	110,314	101,746	7.767%
10	j3022_2	6.1E+13	7.6E+11	1.242%	27,075	23,765	12.225%
11	j3014_1	7.0E+13	5.3E+11	0.766%	70,211	62,561	10.896%
12	j3024_9	8.2E+13	6.8E+09	0.008%	25,970	23,824	8.263%
13	j3048_8	3.4E+14	4.4E+10	0.013%	72,636	67,838	6.606%
14	j3034_2	4.5E+14	9.5E+09	0.002%	8,283	7,097	14.318%
15	j3013_1	6.6E+14	3.8E+11	0.057%	102,511	83,771	18.281%
16	j3016_3	7.8E+14	2.2E+11	0.029%	105,510	95,492	9.495%
17	j3019_9	8.6E+14	1.2E+12	0.144%	8,864	7,280	17.870%
18	j3041_2	1.1E+15	3.0E+11	0.028%	53,603	46,743	12.798%
19	j3020_4	1.1E+15	1.1E+13	1.004%	9,499	8,445	11.096%
20	j3027_7	1.9E+15	1.3E+12	0.065%	54,870	48,316	11.945%
21	j3017_5	3.1E+15	1.8E+11	0.006%	10,692	9,458	11.541%
22	j3043_2	3.2E+15	2.0E+12	0.061%	51,273	45,233	11.780%
23	j3037_2	6.0E+15	4.8E+11	0.008%	25,393	20,041	21.077%
24	j3021_1	9.6E+15	1.5E+13	0.159%	19,863	18,221	8.267%
25	j3029_3	1.1E+16	1.7E+13	0.165%	83,716	72,010	13.983%
26	j3022_1	1.1E+16	3.5E+11	0.003%	27,189	21,909	19.420%
27	j3017_4	1.3E+16	9.5E+12	0.076%	10,867	8,453	22.214%
28	j3044_7	1.4E+16	3.9E+11	0.003%	57,601	53,111	7.795%
29	j3033_3	1.4E+16	3.0E+12	0.021%	9,518	8,532	10.359%
30	j3044_2	1.6E+16	7.9E+11	0.005%	59,728	56,040	6.175%
31	j301_6	1.8E+16	5.6E+12	0.031%	7,470	6,290	15.797%
32	j3021_9	2.1E+16	5.7E+12	0.027%	32,996	30,706	6.940%
33	j306_4	2.5E+16	4.8E+12	0.019%	41,622	37,950	8.822%
34	j3046_7	2.7E+16	2.2E+13	0.082%	74,803	66,713	10.815%
35	j3038_8	2.7E+16	2.5E+12	0.009%	26,614	25,314	4.885%
36	j301_10	4.4E+16	3.4E+13	0.079%	7,035	6,403	8.984%
37	j3034_10	4.6E+16	2.8E+11	0.001%	6,894	5,774	16.246%

38	j3013_8	5.4E+16	2.1E+13	0.040%	109,543	102,449	6.476%
39	j3042_4	5.7E+16	1.1E+13	0.019%	45,827	41,363	9.741%
40	j3039_9	6.2E+16	2.5E+13	0.041%	25,512	21,852	14.346%
41	j3046_10	7.6E+16	2.2E+13	0.029%	74,004	69,040	6.708%
42	j3029_7	8.3E+16	1.2E+13	0.014%	87,787	75,801	13.654%
43	j306_6	1.4E+17	2.2E+13	0.016%	35,558	25,546	28.157%
44	j3033_8	1.5E+17	2.3E+13	0.015%	9,658	7,610	21.205%
45	j3031_5	1.8E+17	5.4E+12	0.003%	61,749	58,775	4.816%
46	j3033_5	1.9E+17	1.2E+13	0.006%	11,776	9,520	19.158%
47	j307_5	2.0E+17	8.7E+12	0.004%	35,221	31,411	10.817%
48	j3036_7	3.1E+17	2.9E+13	0.009%	9,773	7,399	24.291%
49	j3024_8	6.2E+17	8.5E+12	0.001%	31,737	26,475	16.580%
50	j3043_6	1.5E+18	1.2E+14	0.008%	42,110	35,950	14.628%

6. DISCUSSION OF RESULTS

The results obtained from the experimentation have been compared with the only benchmark published up to now for the *Adaptive Genetic Algorithm* (AGA hereafter) (Ponz-Tienda, et al., 2013). They are analyzed in order to obtain correlations that provide an explanation about the differences observed between similar instances "a priori" (Fig 13). The results obtained, comparing the Parallel B&B and the AGA algorithms, are presented in Table 4.

Table 4 indicates that instances with an initial level of complexity equal to or lower than 10^{14} are not improved compared with the AGA benchmarking. The results of the experimentation are summarized in Table 5, presenting the average improvement of the AGA vs the Parallel B&B, with the number and percentage of improved instances. Table 4 shows that metaheuristics, as AGA, provide excellent results expending less computational time than exact methods. However, metaheuristics methods converge fast, easily relapsing into local optimum as with problem #48 (j3036_7), obtaining results very far away from the optimal value; in this case, it is necessary a benchmarking test to prove the capability of heuristics and metaheuristics methods in order to escape from local optimum.

Table 4
Results comparison between AGA and Parallel B&B

	Instance	*Sum of squares*			*Improvement*		
		Initial	*Levelled*				
			AGA	*Parallel B&B*	*AGA*	*Parallel B&B*	*Improved*
1	j3029_6	110,265	107,109	107,109	2.862%	2.862%	No
2	j3041_1	45,832	41,878	41,878	8.627%	8.627%	No
3	j3025_2	39,398	37,290	37,178	5.351%	5.635%	Yes
4	j3044_4	51,951	49,379	49,379	4.951%	4.951%	No
5	j3037_6	22,127	19,399	19,399	12.329%	12.329%	No
6	j3047_6	76,849	71,273	71,273	7.256%	7.256%	No
7	j3025_9	42,302	40,714	40,714	3.754%	3.754%	No
8	j3032_7	75,573	66,557	66,411	11.930%	12.123%	Yes
9	j3030_10	110,314	101,766	101,746	7.749%	7.767%	Yes
10	j3022_2	27,075	23,765	23,765	12.225%	12.225%	No
11	j3014_1	70,211	62,561	62,561	10.896%	10.896%	No
12	j3024_9	25,970	23,864	23,824	8.109%	8.263%	Yes
13	j3048_8	72,636	67,838	67,838	6.606%	6.606%	No
14	j3034_2	8,283	7,097	7,097	14.318%	14.318%	No
15	j3013_1	102,511	83,869	83,771	18.185%	18.281%	Yes
16	j3016_3	105,510	95,502	95,492	9.485%	9.495%	Yes
17	j3019_9	8,864	7,280	7,280	17.870%	17.870%	No
18	j3041_2	53,603	46,857	46,743	12.585%	12.798%	Yes
19	j3020_4	9,499	8,455	8,445	10.991%	11.096%	Yes
20	j3027_7	54,870	48,316	48,316	11.945%	11.945%	No
21	j3017_5	10,692	9,468	9,458	11.448%	11.541%	Yes
22	j3043_2	51,273	45,233	45,233	11.780%	11.780%	No
23	j3037_2	25,393	20,099	20,041	20.848%	21.077%	Yes
24	j3021_1	19,863	18,221	18,221	8.267%	8.267%	No

25	j3029_3	83,716	72,016	72,010	13.976%	13.983%	Yes
26	j3022_1	27,189	21,929	21,909	19.346%	19.420%	Yes
27	j3017_4	10,867	8,453	8,453	22.214%	22.214%	No
28	j3044_7	57,601	53,111	53,111	7.795%	7.795%	No
29	j3033_3	9,518	8,532	8,532	10.359%	10.359%	No
30	j3044_2	59,728	56,040	56,040	6.175%	6.175%	No
31	j301_6	7,470	6,290	6,290	15.797%	15.797%	No
32	j3021_9	32,996	30,714	30,706	6.916%	6.940%	Yes
33	j306_4	41,622	38,126	37,950	8.399%	8.822%	Yes
34	j3046_7	74,803	66,803	66,713	10.695%	10.815%	Yes
35	j3038_8	26,614	25,382	25,314	4.629%	4.885%	Yes
36	j301_10	7,035	6,427	6,403	8.643%	8.984%	Yes
37	j3034_10	6,894	5,774	5,774	16.246%	16.246%	No
38	j3013_8	109,543	102,709	102,449	6.239%	6.476%	Yes
39	j3042_4	45,827	41,381	41,363	9.702%	9.741%	Yes
40	j3039_9	25,512	21,872	21,852	14.268%	14.346%	Yes
41	j3046_10	74,004	69,040	69,040	6.708%	6.708%	No
42	j3029_7	87,787	75,977	75,801	13.453%	13.654%	Yes
43	j306_6	35,558	25,602	25,546	27.999%	28.157%	Yes
44	j3033_8	9,658	7,618	7,610	21.122%	21.205%	Yes
45	j3031_5	61,749	58,817	58,775	4.748%	4.816%	Yes
46	j3033_5	11,776	9,570	9,520	18.733%	19.158%	Yes
47	j307_5	35,221	31,411	31,411	10.817%	10.817%	No
48	j3036_7	9,773	8,383	7,399	14.223%	24.291%	Yes
49	j3024_8	31,737	26,475	26,475	16.580%	16.580%	No
50	j3043_6	42,110	35,950	35,950	14.628%	14.628%	No

Additionally, as shown in Figure 14, the efficiency grows as the initial complexity increases, providing a covariance $cov(x,y) = -0.604$; this result indicates a strong correlation between efficiency and complexity (note that lower values represent greater efficiency). Nevertheless, this experimentation evidences that low complexity problems are not always easier to solve than bigger instances; in fact, for the same initial complexity there might be a difference of up to three magnitude orders in the number of iteration needed to solve the instance (Fig. 14).

Table 5

Summary of benchmark comparison

Average Improvement		*Improved instances*	
AGA	*Parallel B&B*	*Number*	*%*
11.616%	11.895%	26 of 50	52.00%

The efficiency has been correlated (Table 7, Figures 16, 17 and 18) with several parameters (network-based and resource-based) found in the literature, as explained now. The network-based parameters used were the *Network Complexity (NC)* and the *Flexibility Ratio (FR)* (equivalent to *Order Strength (OS)* or *Graph Density (GD)). The Complexity Index* (*CI*) was not included on the network-based parameters, because the real complexity variability cannot be explained by this parameter as was demonstrated by De Reyck and Herroelen (1996). Most of resource-based parameters take into account resources availability, but since the RLP formulation assumes unlimited resources, including them makes no sense. Particularly for that reason, the parameters *Resource Strength* (Kolisch & Sprecher, 1996) and *Resource Contrainedness* were not included on this analysis; nevertheless, the authors have analyzed the correlation with the *Resource Factor (RF)* (Kolisch & Sprecher, 1996) as it is considered in the literature (Li & Demeulemeester, 2015). Additionally, other parameters not found in the analyzed literature were considered: the *Number of Paths (#P)*, the *Average Total Float (ATF)*, the *Average Free Float (ATF)*, and the *Free Float Complexity (FFC)* (computed as the product of the free float plus one of all the activities).

The improvement sectored by level of complexity shows that AGA provides poorer results, even though complexity increases (see Table 6).

Table 6

Improvement sectored by level of complexity (B&B vs AGA)

Level of complexity	*# Instances*		*% Improved by B&B*	
	By level	*Improved by B&B*	*By level*	*Cumulated*
e08	1	0	0.00%	0.00%
e09	1	0	0.00%	0.00%
e11	1	1	100.00%	33.33%
e12	4	0	0.00%	14.29%
e13	5	3	60.00%	33.33%
e14	5	2	40.00%	35.29%
e15	7	4	57.14%	41.67%
e16	18	11	61.11%	50.00%
e17	7	5	71.43%	53.06%
e18	1	0	0.00%	52.00%
Total	50	26		

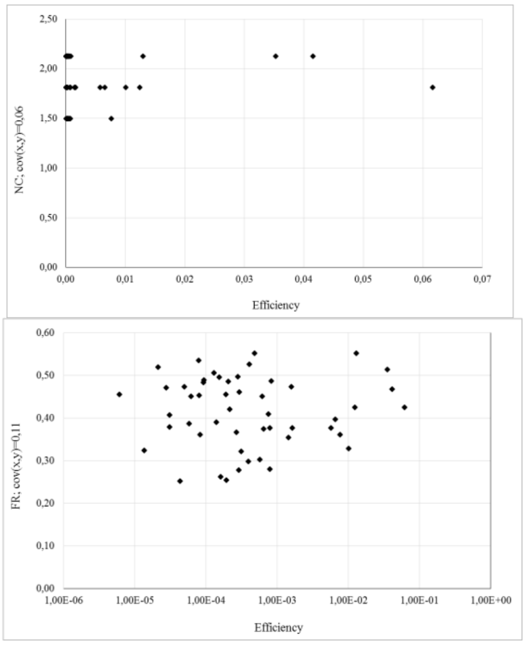


Figure 16 Correlations with network-based parameters

Table 7

Correlations with graph and resource parameters

Network-based parameters		*Resource-based parameter*	*Other parameters*			
NC	*RT*	*RF*	*#P*	*ATF*	*FFC*	*AFF*
0.06	0.11	-0.56	0.05	- 0.22	- 0.05	-0.06

The values observed in Table 7 and represented in Figures 16 to 18 do not provide clear evidences about correlations, with the exceptions of the poor correlation of the efficiency with the *Average Total Float (ATF)* and the correlation with the *Resource Factor (RF)*. The first correlation (Figure 18 upper right hand side) is due to the fact that the efficiency is computed as the relationship between the real and initial complexity, but it does not totally explain the behavior of the variability. The correlation with the *Resource Factor (RF)* (Figure 17) is not conclusive because the PSPLIB is built using the RF as initial parameter with only four values ($RF = \{0.25, 0.50, 0.75\,, 1.00\}$) (Kolisch & Sprecher, 1996), and consequently the correlation may be slanted by the randomness of the selected problems. Nerveless, it suggests that the variability of the efficiency is reduced when the resource factor grows, but this is not a conclusive result. Therefore, more experiments are required to reach more robust conclusions.

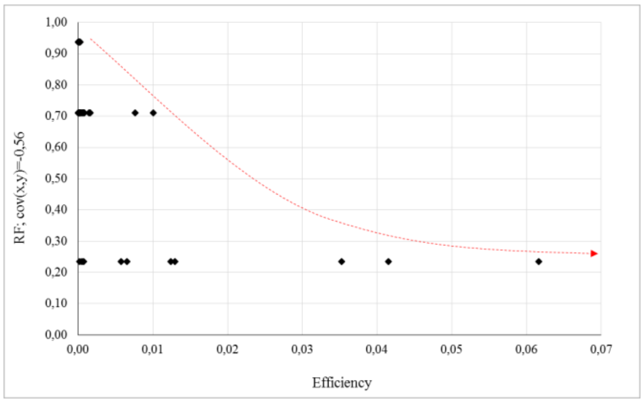


Figure 17 Correlations with Resource Factor (RF)

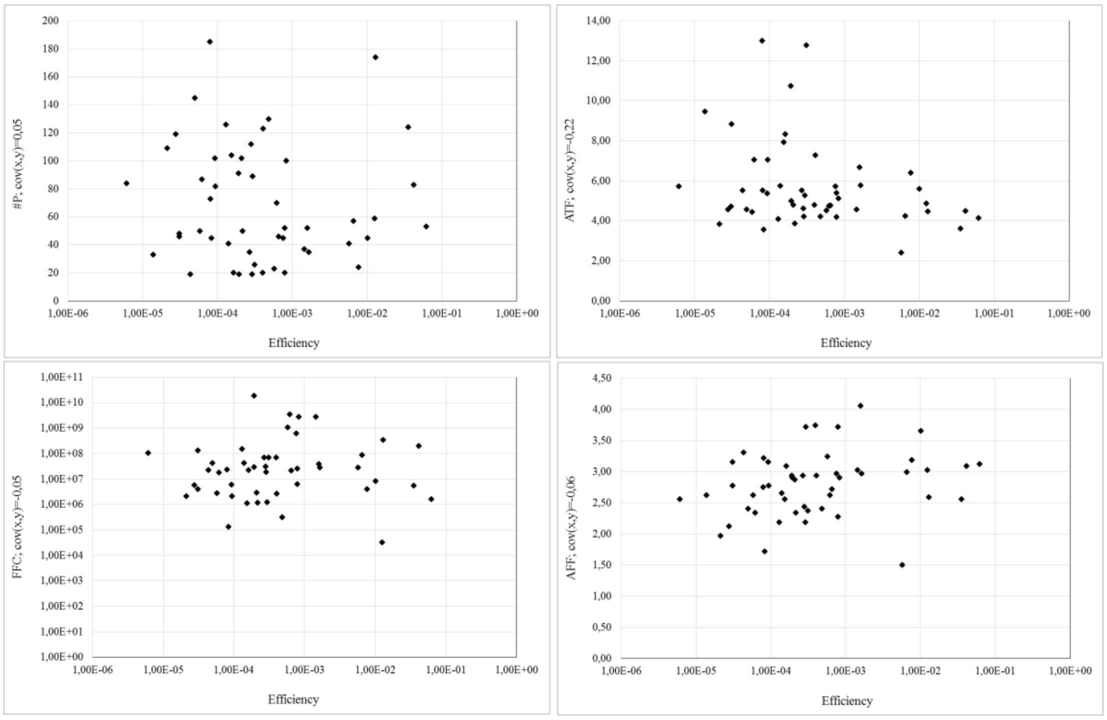


Figure 18 Correlations with other parameters

7. EXAMPLE OF APPLICATION

Besides the benchmarking test described previously, a building project of 15 floors (three underground and 12 aboveground) is used to illustrate the versatility and adaptability of the proposed Parallel *Branch and Bound* algorithm; this example was already used by Ponz-Tienda, et al. (2015) and it has been slightly modified for this case. The building project is completely solved using 71 construction activities contemplating the widest possible set of conditions: the structure is a process overlapped with the processes of masonry, facades and basements with an additional lag of 3, 2 and 1 weeks respectively for removal of formwork to ensure the proper hardening of the concrete. A total of 13 activities/processes which summarize 71 activities and sub-processes are considered. The durations, relationships, weekly resource demand and continuity conditions of each task and process are shown in Table 8.

Table 8
Example of application; durations, relationships and construction constraints.

Activity Code	*Activity description*	*# of subactivities*	*Duration in weeks*	*Weekly resource demand*	*Continuity condition*	*Precedence Relationships* #1	#2
1	Previous works	1	1	5	Yes	-	-
2	Excavations 0.0/-1.0	1	2	3	Yes	$FS_{1-2}(0)$	-
3	Diaphragm-wall	1	8	5	Yes	$FS_{2-3}(0)$	-
4	Excavations	1	6	5	Yes	$FS_{3-4}(0)$	-
5	Rebars for foundation works	1	3	10	Yes	$FS_{4-5}(0)$	-
6	Concrete foundation	1	1	5	Yes	$FS_{4-5}(0)$	-
7	Structure 1 to 15	15	2	15	No	$FS_{6-7}(0)$	-
8	Masonry works 1 to 12	12	1	5	No	$Fl_{7-8}(5,1,3)$	-
9	Facades 1 to 12	12	2	10	No	$Fl_{8-9}(6,1,2)$	
10	Paving works 1 to 12	12	1	5	No	$Fl_{8-9}(1,1,1)$	
11	Office works 1 to 12	12	2	10	No	$Fl_{9-11}(3,1,0)$	$Fl_{10-11}(1,1,0)$
12	Reworks and finishing	1	1	10	Yes	$FS_{11-12}(0)$	-
13	Delivery/reception	1	1	0	Yes	$FS_{12-13}(0)$	-

$FS_{i-j}(z)$ Finish to start relationship with z lag units from activity i to j
$Fl_{i-j}(p_i, p_j, z)$ Flow relationship with z lag units from subactivity p_i to p_j

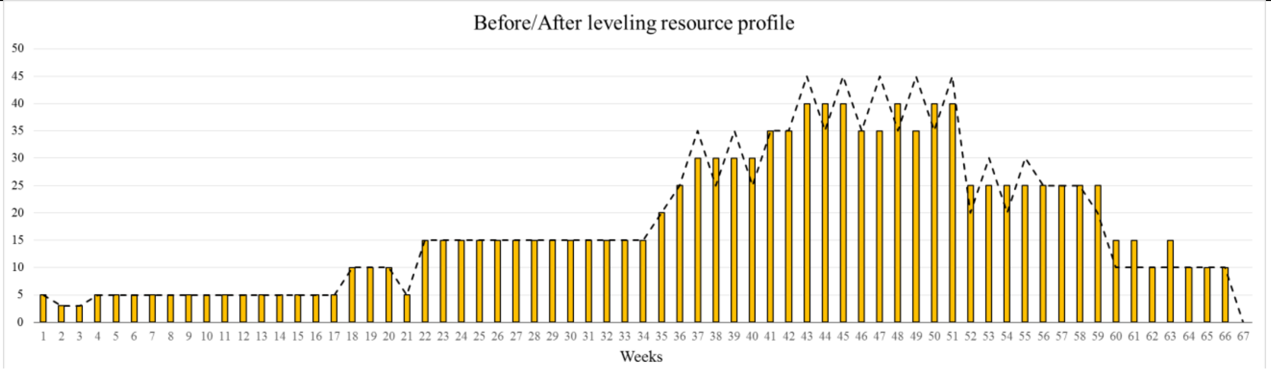


Figure 18 Before/After leveling resource profile of the example of application

Table 9
Example of application; main indicators.

# of problem	*Problem*	*Complexity O(n)* Initial	Real	*Efficiency*	*Sum of squares* Initial	Levelled	*Improvement*
51	EJEM	8.58E+16	1.66E+11	0.00019%	31,418	30,068	4.297%

The construction resource profile of the building project, before and after the leveling process, is presented in Figure 18. In Table 9 the main indicators are displayed, following the criteria used in Table 3.

The building project has been solved to optimality with 160 cores. It has required the generation of 52,467 subgraphs and it has needed 9,869 seconds (2.74 hours). Note that with one core, the required time to solve this problem to optimality would be at least 20 days. This building project is significantly easier and faster to solve than the ones included in the PSPLIB library, processing only the 0.00019% of the sequences (1.66×10^{11}) of the initial complexity of the problem (8.58×10^{16}). The time required to solve this kind of problems is reasonable and practical, considering that unbalanced resources can risk the goals of the construction project; therefore, the use of advanced computer methods in construction projects are justified in order to achieve the best possible schedule.

8. CONCLUSIONS

The efficient use of resources is a key factor in achieving project goals, minimizing resource fluctuations and maximizing results in cost savings by increasing the efficiency of the project sequence. This is particularly significant in construction projects where field operations make fluctuations of resources unproductive and costly. Resource Leveling Problems (RLP) aim to sequence the construction activities that optimize the resource consumption over time, minimizing the variability. Exact algorithms have been proposed by researchers along the literature to offer optimal solutions, but this kind of problems are strongly NP-Hard and, consequently, alternative heuristic and metaheuristic algorithms have been proposed in the literature to find local optimal solutions in an acceptable computational effort. These alternative methods should be tested against exact benchmarks, and not only between them. For this purpose, different libraries have been developed in order to test and benchmark heuristic solutions with the optimal solutions; nevertheless, the PSPLIB for minimal lags is still open to be solved to optimality.

The actual computational capacity allows these problems to be solved to optimality in a reasonable computational time, but such capacity is distributed in individual computers and therefore traditional algorithms must to be redesigned in order to take advantage of these distributed infrastructures. Therefore, the authors have proposed a *Parallel Branch and Bound* algorithm for the Resource Levelling Problem with minimal lags. This algorithm has been implemented gradually in an app developed in C# language using 490 cores in several physical and virtual computers. The algorithm has also been tested with the PSPLIB library.

This way, this proposal is contributing to the body of knowledge of construction project scheduling in the following facets:

- Analyzing the real complexity of the RLP based on the standard parameters found in the literature.
- Proposing a new parallel exact procedure to solve the RLP using an acceptable computational effort considering the scenario of a construction project.
- Providing a benchmarking set of solutions (50 problems of the PSPLIB) to test the goodness of the heuristic algorithms for the RLP.
- Solving to optimality in reasonable computational time a building project with 71 activities, proving the feasibility to be implemented in commercial and professional applications that can help practitioners to schedule real and complex construction projects.

The time required to solve real construction projects is very reasonable and practical (as shown in the previous section), considering that unbalanced resources can risk the goals of the construction project. Therefore, the use of advanced computer methods in construction projects is justified in order to achieve the best possible schedule. Furthermore, the proposed *Parallel Branch and Bound* for the RLP can be easily adapted to solve other regular project scheduling problems as the RCPSP, or combined problems as the RCPSP-RLP, only by including the resource availability as a restriction to the problem and discarding the branches with worst bounds.

This research makes available the optimums of 50 problems (with complexity from 10^{8} to 10^{18}) for the RLP with minimal lags for the first time, allowing contributors to compare their heuristics methods against exact results by obtaining the distance of their solution to the optimal values. It aims to be a benchmark for researchers and a practical tool for practitioners, as the computational capacity increases. This benchmark can be the foundation of future collaborative efforts to estimate the real complexity of the RLP in order to categorize the problems and establish the feasibility to obtain an optimal solution based on the availability of computational resources. Additionally, there are other problems in scheduling as the RCPSP that can take advantage of the *Parallel B&B* algorithm to be solved to optimality; therefore, they can provide new alternatives to the software industry by including the proposed algorithm in their commercial packages.

ACKNOWLEDGEMENTS

This research was supported by the FAPA program of the Universidad de Los Andes (Colombia). The authors would like to thank the research group of Construction Engineering and Management (INgeco), especially to J. S. Rojas-Quintero, and the Department of Systems Engineering at the Universidad de Los Andes. The authors are also grateful to the anonymous reviewers for their valuable and constructive suggestions.

REFERENCES

Adeli, H., 2000. High-performance computing for large-scale analysis, optimization, and control. *Journal of Aerospace Engineering*, 13(1), pp. 1-10.

Adeli, H. & Kamal, O., 1989. Parallel Structural Analysis Using Threads. *Computer-Aided Civil and Infrastructure Engineering*, 4(2), pp. 133-147.

Adeli, H. & Kamal, O., 1992. Concurrent analysis of large structures—II. applications. *Computers & Structures,* 42(3), pp. 425-432.

Adeli, H. & Kamal, O., 2003. *Parallel processing in structural engineering.* s.l.:Taylor & Francis.

Adeli, H., Kamat, M. P., Kulkarni, G. & Vanluchene, R. D., 1993. High-performance computing in structural mechanics and engineering. *Journal of Aerospace Engineering,* 6(3), pp. 249-267.

Adeli, H. & Karim, A., 1997. Scheduling/cost optimization and neural dynamics model for construction. *Journal of Construction Engineering and Management,* 123(4), pp. 450-458..

Adeli, H. & Karim, A., 2001. Construction Scheduling, Cost Optimization, and Management: A New Model Based on Neurocomputing and Object Technologies. In: s.l.:Spon, pp. 233-234.

Adeli, H. & Kumar, S., 1995. Concurrent Structural Optimization on Massively Parallel Supercomputer. *Journal of Structural Engineering,* 121(11), pp. 1588-1597.

Adeli, H. & Kumar, S., 1999. *Distributed Computer-Aided Engineering: For Analysis, Design, and Visualization.* s.l.:CRC Press, Inc. Boca Raton, FL, USA.

Adeli, H. & Vishnubhotla, P., 1987. Parallel Processing. *Computer-Aided Civil and Infrastructure Engineering,* 2(3), pp. 257-269.

Adeli, H. & Wu, M., 1998. Regularization neural network for construction cost estimation. *Journal of Construction Engineering and Management,* 124(1), pp. 18-24.

Ahuja, H., 1976. *Construction Performance Control by Networks.* New York: John Wiley and Sons.

Alsayegh, H. & Hariga, M., 2012. Hybrid meta-heuristic methods for the multi-resource leveling problem with activity splitting. *Automation in Construction,* Volume 27, pp. 89-98.

Anagnostopoulos, K. & Koulinas, G., 2011. Resource-constrained critical path scheduling by a GRASP-based hyperheuristic. *Journal of Computing in Civil Engineering, 26(2), 204-213.,* 26(2), pp. 204-213.

Anagnostopoulos, K. P. & Koulinas, G. K., 2010. A simulated annealing hyperheuristic for construction resource levelling. *Construction Management and Economics,* 28(2), pp. 163-175.

Arditi, D. & Bentotage, S. N., 1996. System for scheduling highway construction projects. *Computer-Aided Civil and Infrastructure Engineering,* 11(2), pp. 123-139.

Bandelloni, M., Tucci, M. & Rinaldi, R., 1994. Optimal Resource Leveling using non-serial dynamic programming,. *European Journal of Operations Research,* Volume 78, pp. 162-178.

Benjaoran, V., Tabyang, W. & Sooksil, N., 2015. Precedence relationship options for the resource levelling problem using a genetic algorithm. *Construction Management and Economics,* 33(9), pp. 711-723.

Bianco, L., Caramia, M. & Giordani, S., 2016. Resource levelling in project scheduling with generalized precedence relationships and variable execution intensities. *OR Spectrum,* 38(2), pp. 405-425.

Burguess, A. & Killebrew, J., 1962. Variation in activity level on a cyclic arrow diagram. *Journal of Industrial Engineering,* 13(2), p. 76–83..

Chakroun, I. & Melab, N., 2015. Towards a heterogeneous and adaptive parallel Branch-and-Bound algorithm. *Journal of Computer and System Sciences,* 81(1), pp. 72-84.

Christodoulou, S., Ellinas, G. & Michaelidou-Kamenou, A., 2009. Minimum moment method for resource leveling using entropymaximization. *Journal of construction engineering and management,* 136(5), pp. 518-527.

Clausen, J. & Perregaard, M., 1999. On the best search strategy in parallel branch-and-bound: Best-First Search versus Lazy Depth-First Search. *Annals of Operations Research,* pp. 1-17.

Coughlan, E. T., Lübbecke, M. E. & Schulz, J., 2010. *A Branch-and-Price Algorithm for Multi-mode Resource Leveling.* Berlin Heidelberg: Springer.

Coughlan, E. T., Lübbecke, M. E. & Schulz, j., 2013. A Branch-Price-and-Cut Algorithm for Multi-Mode Resource Leveling. *Preprint submitted to Elsevier.*

Crainic, T. G., Le Cun, B. & Roucairol, C., 2006. Parallel Branch-and-Bound Algorithms. In: *Parallel Combinatorial Optimization.* s.l.:John Wiley & Sons, Inc., pp. 1-28.

Damci, A., Arditi, D. & Polat, G., 2013a. Resource Leveling in Line-of-Balance Scheduling. *Computer-Aided Civil and Infrastructure Engineering,* Volume 28, p. 679–692.

Damci, A., Arditi, D. & Polat, G., 2013b. Multiresource leveling in line-of-balance scheduling. *Journal of Construction Engineering and Management,* 139(9), pp. 1108-1116..

Damci, A., Arditi, D. & Polat, G., 2016. Impacts of different objective functions on resource leveling in Line-of-Balance scheduling. *KSCE Journal of Civil Engineering,* 20(1), p. 58.67.

De Reyck,, B. & Herroelen, W., 1996. On the use of the complexity index as a measure of complexity in. *European Journal of Operational Research,* Volume 91, pp. 347-366.

Doulabi, S., Seifi, A. & Shariat, S. Y., 2010. Efficient hybrid genetic algorithm for resource leveling via activity splitting. *Journal of Construction Engineering and Management,* 137(2), pp. 137-146.

Drexl, A. & Kimms, A., 2001. Optimization guided lower and upper bounds for the resource investment problem. *The Journal of the Operational Research Society,* 52(3), pp. 340-351.

Easa, S., 1989. Resource leveling in construction by optimization. *Journal of Construction Engineering and Management,* 115(2), pp. 302-316.

El-Rayes, K. & Jun, D., 2009. Optimizing resource leveling in construction projects. *Journal of Construction Engineering and Management,* 135(11), pp. 1172-1180.

Faghihi, V., Nejat, A., Reinschmidt, K. F. & Kang, J. H., 2016. Automation in construction scheduling: a review of the literature. *The International Journal of Advanced Manufacturing Technology,* 81(9), pp. 1845-1856.

Florez, L., Castro-Lacouture, D. & Medaglia, A. L., 2012. Sustainable workforce scheduling in construction program management. *Journal of the Operational Research Society,* 64(8), pp. 1169-1181.

Gaitanidis, A., Vassiliadis, V., Kyriklidis, C. & Dounias, G., 2016. *Hybrid Evolutionary Algorithms in Resource Leveling Optimization: Application in a Large Real Construction Project of a 50000 DWT Ship.* Thessaloniki, Greece, ACM, p. Article No. 25.

Gather, T., Zimmermann, J. & Bartels, J.-H., 2011. Exact methods for the resource levelling problem. *Journal of Scheduling,* 14(6), pp. 557-569.

Georgy, M. E., 2008. Evolutionary resource scheduler for linear projects. *Automation in Construction,* 17(5), pp. 573-583.

Hariga, M. & El-Sayegh, S. M., 2010. Cost optimization model for the multiresource leveling problem with allowed activity splitting. *Journal of Construction Engineering and Management,* 137(1), pp. 56-64.

Harris, R., 1978. *Precedence and Arrow Networking Techniques for Construction.* s.l.:Wiley.

Harris, R. B., 1990. Packing method for resource leveling (PACK). *Journal of Construction Engineering and Management,* 116(2), pp. 331-350.

Hegazy, T., 1999. Optimization of resource allocation and leveling using genetic algorithms. *Journal of construction engineering and management,* 125(3), pp. 167-175.

Heon Jun, D. & El-Rayes, K., 2011. Multiobjective optimization of resource leveling and allocation during construction scheduling. *Journal of Construction Engineering and Management,* 137(12), pp. 1080-1088.

Hinze, J., 2012. *Construction Planning and Scheduling (4th Ed.).* s.l.:Prentice Hall, Upper Saddle River (NJ).

Hiyassat, M. A. S., 2000. Modification of minimum moment approach in resource leveling. Journal of Construction Engineering and Management. 126(4), pp. 278-284.

Hiyassat, M. A. S., 2001. Applying modified minimum moment method to multiple resource leveling. *Journal of Construction Engineering and Management,* 127(3), 192-198.(3), pp. 192-198.

Hossein Hashemi Doulabi, S., Seifi, A. & Shariat, S. Y., 2010. Efficient hybrid genetic algorithm for resource leveling via activity splitting. *Journal of Construction Engineering and Management,* 137(2), pp. 137-146.

Hu, J. & Flood, I., 2012. *A multi-objective scheduling model for solving the resource-constrained project scheduling and resource leveling problems.* s.l., s.n., pp. 49-56.

Ismail, M. M., El-Raoof, O. & El-Wahed, W. F., 2014. A Parallel Branch and Bound Algorithm for Solving Large Scale Integer Programming Problems. *Applied Mathematics & Information Sciences,* 8(4), pp. 1691-1698.

Jun, D. & El-Rayes, K., 2011. Multiobjective optimization of resource leveling and allocation during construction scheduling. *Journal of Construction Engineering and Management,* 137(12), pp. 1080-1088.

Kolisch, R., Schwindt, C. & Sprecher, A., 1999. Benchmark Instances for Project Scheduling Problems. In: *Project Scheduling; Recent Models, Algorithms and Applications.* Boston: Kluwer Academic Publishers, pp. 197-212.

Kolisch, R. & Sprecher, A., 1996. PSPLIB-a project scheduling problem library. *European journal of operational research,* 96(1), pp. 205-216.

Koulinas, G. & Anagnostopoulos, K., 2013. A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities. *Automation in Construction,* Volume 31, pp. 169-175.

Lai, T.-H. & Sahni, S., 1984. Anomalies in parallel branch-and-bound algorithms. *Communications of the ACM,* 27(6), pp. 594-602.

Leu, S. S., Yang, C. H. & Huang, J. C., 2000. Resource leveling in construction by genetic algorithm-based optimization and its decision support system application. *Automation in construction,* 10(1), pp. 27-41.

Li, H. & Demeulemeester, E., 2015. A genetic algorithm for the robust resource leveling problem.. *Journal of Scheduling,* pp. 1-18.

Li, H. X. Z. & Demeulemeester, E., 2014. Scheduling policies for the stochastic resource leveling problem. *Journal of Construction Engineering and Management,* 141(2), p. 04014072.

Lim, T. K., Yi, C. Y., Lee, D. E. & Arditi, D., 2014. Concurrent construction scheduling simulation algorithm. *Computer-Aided Civil and Infrastructure Engineering,* 29(6), pp. 449-463.

McKeown, G. P., Rayward-Smith, V. J. & S. A. Rush, H. J., 1991. *Using transputer network to solve B&B problems*. s.l., 2, p. 781–800.

Menesi, W. & Hegazy, T., 2014. Multimode resource-constrained scheduling and leveling for practical-size projects. *Journal of management in engineering,* 31(6), p. 04014092.

Neumann, K., Schwindt, C. & Zimmermann, J., 2003. *Project Scheduling with Time Windows and Scarce Resources*. Berling: Springer.

Neumann, K. & Zimmermann, J., 1999. Methods for Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions and Schedule-Dependent Time Windows. In: *Project Scheduling*. s.l.:Springer, pp. 261-287.

Neumann, K. & Zimmermann, J., 2000. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research,* 127(2), pp. 425-443.

Nübel, H., 2001. The resource renting problem subject to temporal constraints. *OR Spektrum,* Volume 23, pp. 359-381.

Perregaard, M. & Clausen, J., 1998. Parallel branch-and-bound methods for thejob-shop scheduling problem. *Annals of Operations Research,* Volume 83, pp. 137-160.

Petrovic, R., 1969. On optimization of resource leveling in project plans. In: *Project planning by network analysis*. North-Holland, Amsterdam,: Lombaers HJ, pp. 268-273.

Ponz-Tienda, J. L., Pellicer, E., Benlloch-Marco, J. & Andrés-Romano, C., 2015. The Fuzzy Project Scheduling Problem with Minimal Generalized Precedence Relations. *Computer-Aided Civil and Infrastructure Engineering,* 30(11), pp. 872-891.

Ponz-Tienda, J., Yepes, V., Pellicer, E. & Moreno-Flores, J., 2013. The Resource Leveling Problem with multiple resources using an adaptive. *Automation in Construction,* Volume 29, pp. 161-172.

Pritsker, A. A. B., Waiters, L. J. & Wolfe, P. M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. *Management science,* 16(1), pp. 93-108.

Ranjbar, M., 2013. A path-relinking metaheuristic for the resource levelling problem. *Journal of the Operational Research Society*, 64(7), pp. 1071-1078..

Rieck, J. & Zimmermann, J., 2015. Exact Methods for Resource Leveling Problems. In: *Handbook on Project Management and Scheduling Vol. 1.* s.l.:Springer International Publishing, pp. 361-387.

Rieck, J., Zimmermann, J. & Gather, T., 2012. Mixed-integer linear programming for resource leveling problems. *European Journal of Operational Research,* Volume 221, pp. 27-37.

Rosales, E., Castro, H. & Villamizar, M., 2011. *Unacloud: Opportunistic cloud computing infrastructure as a service*. Rome, Italy, IARIA, pp. 187-194.

Saleh, A. & Adeli, H., 1994a. Microtasking, Macrotasking, and Autotasking for Structural Optimization. *Journal of Aerospace Engineering,* 7(2), pp. 156-174.

Saleh, A. & Adeli, H., 1994b. Parallel Algorithms for Integrated Structural/Control Optimization. *Journal of Aerospace Engineering,* 7(3), pp. 297-314.

Son, J. & Mattila, K. G., 2004. Binary resource leveling model: Activity splitting allowed. *Journal of construction engineering and management*, 130(6), pp. 887-894.

Son, J. & Skibniewski, M. J., 1999. Multiheuristic approach for resource leveling problem in construction engineering: Hybrid approach. *Journal of construction engineering and management,* 125(1), pp. 23-31..

Tang, Y., Liu, R. & Sun, Q., 2014. Two-stage scheduling model for resource leveling of linear projects. *Journal of Construction Engineering and Management,* 140(7), p. 04014022.

Wah, B. W., Li, G. J. & Yu, C. F., 1985. Multiprocessing of Combinatorial Search Problems. *IEEE Computer,* 18(6), pp. 93-108.

Yeniocak, H., 2013. *An efficient branch and bound algorithm for the resource leveling problem,* Middle east technical university: Unpublished Thesis; School of natural and applied sciences.

Younis, M. & Saad, B., 1996. Optimal resource leveling of multi-resource projects. *Computers & Industrial Engineering,* 31(1-2), pp. 1-4.