Improved lower bounds for hard Project Scheduling instances

Guilherme Henrique Ismael de Azevedo
Universidade Federal Fluminense
Rua Passo da Pátria, 156, sala 309 – Bloco D, São Domingos, Niterói - RJ
guilhermehen@gmail.com

Artur Alves Pessoa
Universidade Federal Fluminense
Rua Passo da Pátria, 156, sala 309 – Bloco D, São Domingos, Niterói - RJ
artur@producao.uff.br

ABSTRACT

In a Resource Constrained Project Scheduling Problem with generalized precedence restrictions (RCPSP/max), one must schedule a set of activities with known duration satisfying precedence restrictions with variable time lags and respecting the availability of resources. The goal is to minimize the project duration (makespan).

This study presents a method to calculate a lower bound on the makespan based on constraint propagation that considers three relaxations of RCPSP/max. The first and the second relaxations are solved by greedy methods and the last one as a linear programming problem. We also introduce artificial resources and other preprocessing techniques to improve the lower bound quality. We report experiments using 58 open benchmark instances with up to 200 activities where our procedure proved the optimality of 3 known solutions and closed 62% of the optimality gap on the average. It also found better lower bounds for 26 out of 79 instances with 500 activities.

KEYWORDS. Project Scheduling. Resource Constraint. Resource Constraint.

Main area (inform by priority the área of the article because JEMS system makes the classification alfabeticaly)

OC - Combinatorial Optimization

PM - Mathematical Programming

AD & GP - OR in Administration & Production Management

1. Introduction

Optimizing project planning is important in every knowledge area since it can avoid waste of time and resources, especially in great projects like rocket launch, building hydro-electrical plants or oil and gas platforms (PINEDO, 2004). The resource constrained project scheduling problem (RCPSP) is a well-known model for such optimization that is defined as follows. Let $V = \{0, 1, ..., N, N+1\}$ be the set of activities to be scheduled using a set $\Re = \{1, \ldots, m\}$ of resources. Each activity $j \in V$ has an execution time p_j and uses a specific amount r_{ij} of resource *i* at every instant of its processing time. Resource *i* has R_i units available during all the time horizon. The schedule must also respect activity precedence, which are represented by the directed graph $G = (V, A)$. For each $(j, l) \in A$, we use d_{jl} to denote the time lag from activity *j* to *l*. In this case, *l* has to start at least d_{jl} time units after *j* starts and *j* has to start at most $-d_{jl}$ units of time after *l* starts. For the RCPSP/max, nonnegative cycles are allowed in *A*. The activities 0 and $N+1$ represent the beginning and the end of the project, having null execution times and null use of resources. Every activity with no other predecessor will succeed 0, and every activity with no other successor will precede $N+1$. In this study, the objective is to find the minimum completion time for the project, also known as makespan and also defined as the finish time of activity $N+1$.

RCPSP and its variations are NP-hard as they generalize shop problems, like open shop or job shop (BLAZEWICZ ET AL., 1983). Since it is difficult and time costly to prove optimality of solutions for this problem, finding a good quality lower bound (LB) may bring some useful information during the decision making process. There are two approaches in the literature to calculate a LB for the RCPSP: constructive and destructive methods.

Constructive LBs use relaxed formulations of the problem, which are, generally, easier to solve. One of the most used relaxations is the workload based one. For a given resource *i*, the total workload required by each activity *j* is given by $r_{ij} \times p_j$. The workload based relaxation tries to schedule activities so as to satisfy the workload requirements instead of the resource constraints. For example, Koné et al. (2011) introduced a mathematical formulation based on start and finish events. Kooli et al. (2010) presented a Mixed Linear Problem (MIP) that generalizes workload constraints. Franck et al. (2001) also presented a workload-based method using the concepts of Baptiste et al. (1999) and Dorndorf et al. (1999). Schutt et al. (2010) use a finite domain solver in a lazy clause generation approach with a hybrid of finite domain and Boolean satisfiability solving.

Demassey et al. (2005) define the destructive method as a procedure that defines hypothetical upper bounds T' on the makespan aiming to prove that there is no solution respecting this upper bound. If such a proof is successful, one can conclude that $T'+1$ is a lower bound on the optimal makespan. Brucker & Knust (2000) presented a destructive LB that uses a constraint propagation method and a MIP to try to prove that T' is an infeasible upper bound.

The method presented in this paper contains three different workload approaches to be used in both constructive and destructive LBs. The first and the second ones are single resource workload relaxations of RCPSP/max. The first relaxation does not consider the activities execution intervals, but the second does consider. The third method uses a linear programming (LP) relaxation that considers all resources in parallel. At first, the three methods are used in a constructive approach to update execution intervals and the LB on makespan. Once an interval is updated, the information is transmitted to the other intervals through by a precedence constraint propagation method. After the constructive approach finishes, we use the same three relaxations in a destructive approach to update the LB on the makespan.

We used the instances available at PSPLib (2012) as benchmark. As Schutt et al. (2010) proved optimality for a great number of them with 10 to 200 activities in little computational time, we consider only the 58 hard instances that could not be solved by their method. All these instances have available feasible solutions whose optimality is not proved and available lower bounds reported by Franck et al. (2001). For these instances, no LB is provided by Schutt et al. (2010). In the reported experiments, we proved the optimality of 3 feasible solutions for the first time, for instances with 30 activities, and closed 62% of the optimality gaps on the average. We also tested a faster but weaker version of our method on 79 instances with 500 activities, finding better LBs for 26 of them, and closing 5.5% of the optimality gaps on the average.

This paper has the following structure: Section 2 presents our approach to calculate lower bounds on the makespan. Section 3 describes the implementation and Section 4 the tests and computational results. Finally, Section 5 presents our conclusions.

2. Our Approach

Our method maintains an execution interval for each activity defined as an interval where it must be completely executed. The execution interval of activity *j* is denoted by $\left(ES_j, LF_j\right)$, where the Early-Start (ES) is the first instant it can start and the Late-Finish (LF) is the last moment it can end. Although the execution intervals are not in the problem definition, they are implied by in the precedence relations. To initialize the execution intervals, first we consider $\left(ES_j, LF_j\right) = \left(0, T\right)$ for all $j \in V$, where T is a valid upper bound (UB). Then we use the propagation method, described below, to check and update them if possible. This method is divided in two phases, one for ES and other for LF updates.

We propagate the ES updates as follows. Let us consider that ES_j was updated. If there is a precedence arc $\left(j,l\right) \in A$ that does not respect the constraint $ES_j + d_{jl} \geq ES_l$, we update ES_l to $ES_j + d_{jl}$. This check must be repeated until every updated ES_j is tested and no more updates are performed. Similarly, we propagate LF updates as follows. Whenever LF_j is updated, if the constraint $LF_l \leq LF_j - d_{lj}$ is not satisfied, due to $\left(l, j\right) \in A$, we update LF_l to $LF_j - d_{lj}$. This check must also be repeated until every updated LF_j is tested and no more updates are performed.

Until this point, the execution intervals consider only precedence relations. In the next subsections, we will present the three workload methods to update the intervals considering also the resource constraints, each one based on a different relaxation of the problem. In the first relaxation, we schedule the workload for each resource individually regardless the execution intervals. In the second, the workload is also schedule for each resource individually, but considering execution intervals. In order to increase the quality of the lower bound, we also present a way to include artificial resources and other techniques to strengthen the relaxation. The third method is a domain reduction by LP that considers all resources in parallel, which is stronger but slower than the previous methods. For all the relaxations preemption is allowed.

2.1. Domain reduction by single-resource relaxation

The first method presented to update the execution intervals uses a workload based relaxation that considers a single resource to check if activity $j \in V$ can start at ES_j and finish at LF_j. In order to check the validity of the starting time ES_j for activity *j*, we consider the total workload that must occur before *j* starts. If we prove that it is not possible to schedule all this workload in the interval $\left(0, ES_j\right)$, then we can conclude that ES_j can be increased by at least one time unit. More generally, let α be an arbitrary instant before ES_j. If we prove that it is not possible to schedule the workload that must occur between α and the start of *j* in the interval $\left(\alpha, ES_j\right)$, then we can conclude that ES_j can be increased. Similarly, given an instant β after LF_j, if we find out that it is not possible to schedule all the workload that must occur between the finish of *j* and β in the interval (LF_j, β), then we prove that LF_j can be reduced by at least one time unit.

2.1.1. Workload requirements

Here, we show how to calculate the workload of each $l \in V$ that must occur in (α, ES_j) assuming that *j* starts at ES_j, or in (LF_j, β) assuming that *j* finishes at LF_j. Beside the assumptions, the method to calculate the workload for both intervals is the same. Hence, we will present it for a generic interval (LB, UB) with no assumption. Later, we show how to consider these assumptions.

Let (LB'_l, UB'_l) be the current execution interval for activity $l \in V$ and Δ_l be the part of the processing time of *l* that must occur in the interval (LB, UB). We can calculate Δ_l by the formula $\Delta_l = p_l - \max\{\phi_l ; \Phi_l\}$, where ϕ_l is the part of the execution time of *l* out of (LB, UB) when it is scheduled starting at LB'_l and Φ_l is the part of the execution time of *l* out of (LB, UB) when it is scheduled finishing at UB'_l. ϕ_l and Φ_l can be calculated as follows:

$$\phi_l = \max\{LB - LB'_l ; 0\} + \max\{LB'_l + p_l - UB ; 0\} \quad (1)$$

$$\Phi_l = \max\{UB'_l - UB ; 0\} + \max\{LB - (UB'_l - p_l) ; 0\} \quad (2)$$

Once we calculated Δ_l, the workload of activity *l* that is required in the interval (LB, UB) for the resource *i* is $w_{il} = r_{il} \times \Delta_l$.

When checking whether the activity $j \in V$ can start at ES_j, we do not use the current execution intervals for remaining activities. Instead, we temporarily update such intervals assuming that *j* starts at ES_j and using the propagation method described in the beginning of this section. Then, we calculate the workload of each activity that must occur in (α, ES_j) as shown above (let $LB = \alpha$ and $UB = ES_j$) and use the single resource workload relaxation to check if it is possible to schedule all this workload for each resource individually. Similarly, to check whether $j \in V$ can finish at LF_j we temporarily update the execution intervals assuming *j* finishes at LF_j and calculate the workload that must occur in (LF_j, β) as shown above (let $LB = LF_j$ and $UB = \beta$).

After all checks are performed the execution intervals for all activities other than *j* are restored to their original values regardless of whether the interval of *j* is updated or not. If it is updated, we propagate the information to the other execution intervals.

For a given resource *i*, the total workload to schedule in (α, ES_j) is $\sum_{l \in V \setminus \{j\}} w_{il}$ and the total available work is $R_i(ES_j - \alpha)$. Thus, in order to be possible that $j \in V$ to starts at ES_j, the constraint $\sum_{l \in V \setminus \{j\}} w_{il} \le R_i(ES_j - \alpha)$ must be satisfied. Otherwise, ES_j can be updated to $\alpha + \left(\sum_{l \in V \setminus \{j\}} w_{il} / R_i\right)$. To update LF_j, one could use a similar procedure, but considering the workload in the interval (LF_j, β) and the constraint $\sum_{l \in V \setminus \{j\}} w_{il} \le R_i(\beta - LF_j)$. In this case, if the constraint is not satisfied, LF_j is updated to $\beta - \left(\sum_{l \in V \setminus \{j\}} w_{il} / R_i\right)$.

We also call the update criterion described above "*single-resource relaxation with no execution interval*", as it does not consider the execution interval of any activity for scheduling workloads. In the next subsection, we define the "*single-resource relaxation considering the execution intervals*" and present a method to solve it. This method generalizes the ideas introduced above to satisfy the interval workload constraints. The resulting relaxation is stronger but harder to solve than the one with no interval. We use both relaxations in our method.

2.1.2. Single-resource workload relaxation considering execution intervals

The single-resource workload problem (SRWP) can be described as follows. Let $V' = \{1, 2, ..., N'\}$ be a set of activities to be scheduled to a single resource during the time horizon $(0;T')$ and allowing preemption. Each $k \in V'$ has a workload w'_k to be scheduled during the time interval (ES'_k, LF'_k). There are H subintervals of the time horizon, each subinterval h starting at UB_{h-1} and finishing at UB_h, for $h \in \{1,\dots,H\}$. For every activity $k \in V'$, there are subintervals h' and h'' such that $UB_{h'} = ES_j$ and $UB_{h''} = LF_j$. The time horizon bounds are such that $UB_0 = 0$ and $UB_H = T'$. There is a single resource with availability W'_h on each subinterval h of the time horizon. The objective is to find the amount ω_{kh} of workload for each activity k to be scheduled in each subinterval h such that $\sum_{k\in V'} \omega_{kh} \le W'_h$ for all $h \in \{1,\dots,H\}$ and $\sum_{h=1}^{H} \omega_{kh} = w'_k$ for all $k \in V'$.

Assume without loss of generality that $LF'_1 \le LF'_2 \le \dots \le LF'_N$. We use a greedy method to solve the SRWP defined above. For every subinterval h from 1 to H, we schedule the activity with lowest index that can be executed in h ($(ES'_k, LF'_k) \cap (UB_{h-1}, UB_h) \ne \emptyset$) and whose workload is not completely scheduled until $\sum_{k\in V'} \omega_{kh} \le W'_h$ or there is no activity that satisfies the requirements to be scheduled in h. At the end, if every activity is completely scheduled, i.e. if the constraint $\sum_{u=1}^{H} \omega_{ku} = w'_k$ is satisfied for every $k \in V'$, then the SRWP is feasible. Otherwise, it is infeasible. Next, we prove that the previous procedure is correct.

Lemma 1: *Let S_1 be a feasible solution for the SRSP where there is at least one pair of activities $j, l \in V'$ such that $j < l$ and a part of l is scheduled in a subinterval before a part of j when both activities could be scheduled. It is always possible to change S_1 so that the workload of l is scheduled in the same subinterval as j or after, without changing the schedule for the other activities.*

Proof: In S_1, if $\sum_{k\in V} \omega_{kh} < W'_h$ then it is possible to transfer part of $\omega_{jh'}$ to h so that ω_{jh} and ω_{lh} are in the same subinterval. If there is still a part of j in h' then the new $\omega_{jh'}$ and ω_{lh} are in one of the following cases.

If $\omega_{jh'} = \omega_{lh}$, since both j and l can be scheduled in h and $LF_l \ge LF_j$, then we can change their positions and S_1 is still feasible as the total workload in h and in h' do not change. If $\omega_{jh'} > \omega_{lh}$ then we divide $\omega_{jh'}$ in to $(\omega_{jh'})_1 = \omega_{lh}$ and $(\omega_{jh'})_2 = \omega_{jh'} - (\omega_{jh'})_1$ to perform the change between $(\omega_{jh'})_1$ and ω_{lh}. If $\omega_{jh'} < \omega_{lh}$ then it is possible to divide ω_{lh} in to $(\omega_{lh})_1 = \omega_{jh'}$ and $(\omega_{lh})_2 = \omega_{lh} - (\omega_{lh})_1$ and perform the change between $(\omega_{lh})_1$ and $\omega_{jh'}$. In all cases, after the changes in S_1 the total workload in h and in h' do not change, the scheduling for the other activities' workload do not change and the part of l's workload is in the same interval j's or later, as Lemma 1 describes. □

The explanation previously presented shows that for a given feasible solution it is always possible to change the scheduling order of 2 activities workload, or parts of them, so that the activity with the smallest LF is scheduled before.

Theorem 1: *If the problem is feasible, then it is always possible to find a feasible solution scheduling the activities in the earliest possible time interval, respecting execution intervals and* *workload constraints.*

Proof: We prove it by induction on the value of N'. For $N' = 1$, the theorem obviously holds. Assume that the theorem holds for $N' = k$. We must prove that it also holds for $N' = k+1$. Let $j = 1$ and S_1 be a feasible solution for this problem such that there is at least a part of an $l > j$ scheduled before a part of *j*. By Lemma 1, it is possible to perform all the necessary changes in order to have *j* completely scheduled in the same interval or earlier than all the other activities after ES_j. Thus, if the problem is feasible then the algorithm schedules *j* in a feasible position. For the remaining activities, consider modified problem without *j* where every workload used by *j* is subtracted from the availability of the corresponding intervals. By the inductive hypothesis, since this subproblem has *k* activities, the algorithm finds a feasible solution for it if possible. As a result, it will find a feasible solution to the original problem, when it is feasible. □

2.2. Single-resource workload subproblem

As the original problem has *m* resources and the relaxation has a single one, there will be *m* subproblems to check the ES of each activity, one for each resource, and other *m* subproblems to check the LF of each activity.

To check ES_j for resource *i*, there will be $N' = N$ activities, and each activity $k \in V$ will have an associated $l \in V'$ with $w'_l = r_{ik} \times \Delta_k$ to be scheduled in $\left(ES'_l, LF'_l\right)$, where $ES'_l = \max\{ES_k ; \alpha\}$ and $LF'_l = \min\{LF_k ; ES_j\}$. The work available at the subinterval *h* will be $W'_h = R_i\left(UB_h - UB_{h-1}\right)$. If at least one of the subproblems is infeasible, then ES_j can be updated to $ES_j + 1$. This procedure is used in a binary search to define the new value of ES_j. After that, the information is propagated to the other activities of V.

To check LF_j for the resource i, we use a similar procedure, but the activity intervals are defined as $ES'_l = \max\{ES_k ; LF_j\}$ and. $LF'_l = \min\{LF_k ; \beta\}$ If at least one of the subproblems is infeasible, then LF_j can be updated to $LF_j - 1$.

Although the single-resource relaxation considers the execution intervals, some useful information may be lost because the problem considers only one resource at a time, without precedence relations and the workload constraint allows activities to use more resource units than they really instantly require. In order to avoid it, we developed two additional techniques that we present in the next subsection.

2.3. Artificial resources

The lower bounds obtained through the workload approach can be further improved by the creation of artificial resources. The effectiveness of such technique relies on the fact that the workload relaxation allows for using resources fractionally in such a way that would not be possible in a feasible solution. For instance, if two jobs need two units of a given resource each and there are only three units available, the execution of these two jobs could not overlap in time. On the other hand, the workload relaxation would allow one job and half of the other to be executed simultaneously. In this case, instead of trying to consider the non-overlapping constraint directly in the LB calculation, we represent it as an additional (artificial) resource with one available unit and having one unit required by each job, so that it is taken into account without changing the lower bounding algorithm. The mathematical concept of artificial resources is presented below.

Let r_{fj} be the requirement of activity *j* for the artificial resource *f* and R_f be the availability of this resource during the time horizon of the project. The condition $R_f \geq B$, such that *B* is the optimal value of the MIP (3)-(6), is sufficient to ensure that the new resource does not change the feasible region of the RCPSP.

$$B = \max \sum_{j \in V} r_{fj} \times y_j \tag{3}$$

s.t.:

$$\sum_{j \in V} r_{ij} \times y_j \le R_i \qquad \forall i \in \Re \tag{4}$$

$$y_j + y_k \le 1 \qquad \forall (j,k) \in A, d_{jk} \ge p_j \text{ or } ES_k \ge LF_j \text{ or } ES_j \ge LF_k \tag{5}$$

$$y_j \in \{0,1\} \tag{6}$$

In this MIP, y_j is an auxiliary variable that is 1 if *j* is in the subset of activities that maximizes the use of resource *f*, calculated in equation (3), and 0 otherwise. The chosen subset must respect the resource constraints and their execution intervals must overlap. Those requirements are guaranteed by Equations (4) and (5), respectively.

In the next subsections, we present two specific families kinds of artificial resources to be considered in the workload method. The first is called “clique resources” and the second, “hypergraph clique resources”.

2.3.1. Clique resources

Let $G' = (V, E)$ be an undirected graph where each activity of *V* is a vertex and $\{j,k\} \in E$ if and only if *j* and *k* can not be executed in parallel. A clique is a set of vertices $Q \subseteq V$, such that if $j \ne k$ and $j,k \in Q$ then $\{j,k\} \in E$.

For the construction of G', let $j,k \in V$ be two activities. If for any resource *i*, $r_{ij} + r_{ik} > R_i$, if $(j,k) \in A$ with $d_{jk} > p_j$, if $ES_j > LF_k$ or if $ES_k > LF_j$, then $\{j,k\} \in E$. For each clique *Q*, an artificial resource *f* can be added, with $r_{fj} = R_f$ if $j \in Q$ and $r_{fj} = 0$ if $j \notin Q$. For simplicity, we consider $R_f = 1$.

2.3.2. Hypergraph clique resources

The main idea is to generalize clique resources to consider the cases where 3 activities can not be executed in parallel, but pairs among those 3 may be allowed. Let $H' = (V, E')$ be a hypergraph where each activity of *V* is a vertex and each hyperedge $E_i = \{j,k,l\} \in E'$ if and only if no more than two activities among *j*, *k* and *l* can be executed in parallel. Since each hyperedge in *E* has exactly three endpoints, H' is classified as 3-uniform. Each subset $Q' \subseteq V$ induces a sub-hypergraph $H'' = (Q', E'')$ of H' such that $E'' = \{E_i \in E' \mid E_i \subseteq Q'\}$. A 3-uniform complete hypergraph H'' induced by Q' in H' is a sub hypergraph such that, for all $j,k,l \in Q'$, $\{j,k,l\} \in E''$, i.e. no more than two activities in Q' can be executed in parallel.

For the RCPSP, let the three activities be $j,k,l \in V$, if at least one of the edges $\{j,k\}$, $\{j,l\}$ or $\{k,l\}$ belongs to *E* as described in subsection 2.3.1, or for any resource *i* we have $r_{ij} + r_{ik} + r_{il} > R_i$. For each generalized $Q' \subseteq V$ that induce a 3-uniform complete sub-hypergraph in H', an artificial resource *f* is added with $r_{fj} = R_f$ if $j \in Q'$ and there is no $k \in Q'$ that can be executed simultaneously with *j*. If $j \in Q'$ and there is at least one $k \in Q'$ that can be executed simultaneously with *j*, then $r_{fj} = R_f / 2$. If $j \notin Q'$ then $r_{fj} = 0$. For simplicity, we consider $R_f = 2$.

2.4. Resource availability reduction

The LB obtained through the workload approach can also be improved by reducing the resource availabilities without changing the feasible space when it is possible. For instance, consider a given interval of the project time horizon, whose duration is 2 time units, where three units of a given resource is available, but only two activities can be scheduled each one needing one resource unit, and having a duration of 3 time units. It is clear that, for the original problem, the maximum resource usage at any instant of the given interval will be 2. Notice that, if this information is not available to the workload relaxation, it does not detect that the two activities cannot be completely scheduled in the given interval since the total available work is equal to the sum of their workloads. On the other hand, if we consider that the maximum resource usage is 2, them the total available work will be 4, forcing part of the activities to be scheduled before or after this interval.

In a more general way, let V_t' be the set of activities that can be scheduled in the interval *t*. The maximum usage of the resource *i* can be calculated by the MIP below:

$$\max \sum_{j\in V_t'} r_{ij} \times y_j \tag{7}$$

s.t.:

$$\sum_{j\in V_t'} r_{ij} \times y_j \le R_i \tag{8}$$

$$y_j \in \{0,1\} \qquad \forall j \in V_t' \tag{9}$$

In this MIP, y_j is a binary variable that is 1 if the activity *j* is in the combination that maximizes the use of resource *i* and zero otherwise. One might have noted that the previous problem is exactly the 0-1 knapsack Problem. This problem is also a NP-Hard problem, but it is widely studied in the literature. In this study, we considered the method presented by Psinger (1997) to solve it.

2.5. Domain reduction by linear programming

This method uses a linear programming relaxation to update each activity both ES and LF. The idea is to consider the workload for all the resources for every subinterval of the time horizon instead of considering the resource constraint. We use the relaxation defined below.

Let $V' = \{1, 2, ..., N'\}$ be a set of activities to be scheduled to m' resources during the time horizon $(0;T')$ and $U' = \{0,...,T'\}$ be an ordered set of instants to divide the time horizon in subintervals. Each activity *k* requires r_{ik}' units of the resource *i* and has an execution time p_k' to be scheduled in (ES_k', LF_k'). Preemption is allowed. For every subinterval *h* of the time horizon, the resource *i* has an availability of R_{ih}'. The total workload to *i* in *h* must respect $W_{ih}' = R_{ih}'(UB_h - UB_{h-1})$ the available work. There are *H* subintervals of the time horizon, each subinterval *h* starts at $UB_{h-1} \in U'$ and finishes at $UB_h \in U'$. For every activity $k \in V'$ there are $UB_{h'}, UB_{h''} \in U'$ such that $UB_{h'} = ES_k$ and $UB_{h''} = LF_k$. The objective is to define the maximum part of an activity *l* that can be scheduled in (a,b) for given instants $a,b \in U'$. The problem can be solved as the LP below.

$$\alpha = \max \sum_{h=\Gamma(a)+1}^{\Gamma(b)} x_{lh} \tag{10}$$

s.t.:

$$\sum_{\substack{h=1 \\ k\in V_h'}}^{H} x_{kh} = p_k' \qquad \forall k \in V' \tag{11}$$

$$\sum_{k\in V_h'} r_{ik}' \times x_{kh} \le R_{ih}'(UB_h' - UB_{h-1}') \qquad \forall i \in \Re, h = 1,\dots,H \tag{12}$$

$$0 \le x_{kh} \le \min\{p_k'; UB_h' - UB_{h-1}'\} \qquad \forall h = 1,\dots,H,\ k \in V_h' \tag{13}$$

In this LP, x_{kh} represents the part of the activity *k* that is executed in the interval *h*, V_h' is the set of activities that can execute in *h* and $\Gamma(y)$ denotes the index of the interval that finishes at y. The equation (10) calculates the maximum amount of *l*'s execution time that can be scheduled in (a,b). Equation (11) guaranties that all execution time of every activity is scheduled. Equation (12) makes sure that the workload scheduled for each resource in each subinterval does not exceed the available workload. Equation (13) defines the domain of x_{kh}.

We use this relaxation to define a new value to ES_j as follows. Let us consider $V' = V$, $m' = m$, $(0,T') = (0,T)$, $p'_k = p_k$, $(ES'_k, LF'_k) = (ES_k, LF_k)$, $r'_{ik} = r_{ik}$, $R'_{ih} = R_{ih}$ and $(a,b) = (ES_j, ES_j + p_j)$. For every activity *k*, $ES'_k, ES'_k + p'_k, LF'_k - p'_k, LF'_k \in U'$. If $\alpha < p_j$ then it means that *j* can not be completely scheduled in $(ES_j, ES_j + p_j)$ for the relaxed problem, and so for the original problem. Since the original problem does not allow preemption, *j* can not start before $ES_j + (p_j - \alpha)$, them ES_j can be updated to $ES_j + (p_j - \alpha)$.

To solve the relaxation and update LF_j, we use a similar procedure. The only difference is that $(a,b) = (LF_j - p_j, LF_j)$. If $\alpha < p_j$ then it means that *j* can not finish after $LF_j - (p_j - \alpha)$, them LF_j can be updated to $LF_j - (p_j - \alpha)$.

Since $p'_{N+1} = 0$, it is not possible to update the LB on the makespan using the procedure shown above to update ES_{N+1}. To update ES_{N+1} we use the same problem, but with another objective. The goal of the new problem will be to find the minimum execution time scheduled in (a,b). This problem can be solved as the LP below.

$$\alpha = \min \gamma \tag{14}$$

s.t.:

$$\sum_{\substack{h=1 \\ k \in V'_h}}^{H} x_{kh} = p'_k \qquad \forall k \in V' \tag{11}$$

$$\sum_{k \in V'_h} r'_{ik} \times x_{kh} \le R'_{ih}\left(UB'_h - UB'_{h-1}\right) \qquad \forall i \in \Re, h = 1,\dots,H \tag{12}$$

$$0 \le x_{kh} \le \min\left\{p'_k ; UB'_h - UB'_{h-1}\right\} \qquad \forall h = 1,\dots,H,\ k \in V'_h \tag{13}$$

$$\gamma \ge \sum_{\substack{h=\Gamma(a)+1 \\ k \in V'_h}}^{\Gamma(b)} x_{kh} \qquad \forall k \tag{15}$$

To update the LB on the makespan with this LP we define $(a,b) = (ES_{N+1}, T)$ to schedule all activities using the minimum amount of time after the current lower bound on the makespan. So, if $\gamma > 0$ then we know it is not possible to schedule all activities for the relaxed problem before $ES_{N+1} + \gamma$ and so for the original problem. As a result, the LB on the makespan can be updated to $ES_{N+1} + \gamma$.

We also introduce the following cuts on demand to improve the LB quality, where $\delta(l, h_1, h_2)$ denotes the execution time of *l* that must be scheduled in $(UB_{h_1 - 1}; UB_{h_2})$.

$$\sum_{t=h_1}^{h_2} x_{lt} = \delta(l; h_1; h_2) \tag{16}$$

One might have noted that relaxing the resource constraints to workload ones may result in the same loss of information mentioned in the subsection 2.3. So, in order to avoid some schedules that are not allowed in the original problem, we introduce workload constraints for artificial resources on demand as the constraints of Equation (12). We use the same kinds of artificial resources that were presented in subsections 2.3.1 and 2.3.2.

3. Implementation

The concepts and relaxations presented before were used in two different methods: complete and simplified ones.

The complete method starts initializes the ESs and LFs for every activity using the propagation method presented in Section 2. After that, it adds artificial resources described in 2.3.1 and 2.3.2 until every activity *j* has $r_{fj} \neq 0$ for at least one artificial resource *f*. Next, it applies the workload relaxation with no execution intervals and without resource-availability reduction, as described in 2.1.1 until no more updates are possible.

Then, the greedy method described in 2.1.2 is executed with resource-availability reduction described in 2.4. In the moment that no more updates are possible with the greedy method, we create and solve the LPs described in 2.5, introducing cuts on demand as previously described. Once it is not possible to update any interval using the current LPs, we use the propagation method described in 2. If any ES or LF is updated, then we go back to the greedy method iteration. This is repeated until no more updates are found with both LP and greedy methods.

Until this point, we have used a constructive method that, not only updates the LB on the makespan, but also updates the execution intervals of the activities. Now, in order to further improve the quality of the LB on the makespan, we will use the destructive process, described in Brucker and Knust (1997). From now on, no update will be valid to ESs and LFs. The process defines a T' in (ES_{N+1}, LF_{N+1}) to be tested. Now, we use the two workload-based relaxations previously presented to update activity execution intervals. If at any moment we get $ES_j + p_j > LF_j$ for any $j \in V$ then there is no feasible solution with makespan equal or less than T', i.e. T' is an infeasible value for the makespan. The optimal value for T' in (ES_{N+1}, LF_{N+1}) is found through a binary search procedure.

In order to compare the relaxations alone, we also implemented the procedure considering only the relaxations with single resource. We call this implementation simplified method.

The whole procedure was implemented in C++ and all the tests were made in a 2.13 GHz Intel® Core2Duo® computer with 4 GB of RAM. In order to solve LPs and MIPs, we use the CPLEX 12.

4. Computational tests and results

The methods here presented were tested in the 58 hard instances of the benchmark sets UBO50, UBO100, UBO 200, C, D and J30. The hard instances are the ones that Schutt et al. (2010) could not prove optimality or found a LB. We also solve the 79 benchmark instances of the set UBO500 that are feasible using the only the simplified method. The LBs available at PSPLib (2012) were used as benchmarks to check the quality of our LBs. All instances are available at PSPLib (2012).

Table 1 shows the results obtained by our complete method for the 58 hard instances. In this table the first column represent the set of instances tested, the second represent the number of instances tested in the set, columns “Same LB” and “Better LB” represent the number of instances that our method could find respectively the same and a better LB than the one available in the literature, “Closed” shows the number of instances whose optimality was proved for the first time, “Average Time (s)” shows the average time (in seconds) to solve the instances, “GAP%” represents the average percentage GAP and “% GAP closed” represent the percentage of the GAP closed by our method.

Table 1 – Results for complete method

Group	Instances	Same LB	Better LB	Closed	Average Time (s)	GAP %	% GAP closed
UBO50	3	0	3	0	23.47	9.83%	66.99%
UBO100	10	0	9	0	1,358.88	16.86%	24.40%
UBO200	18	0	15	0	12,176.40	24.32%	10.05%
C	17	5	4	0	1,574.68	16.29%	-4.78%
D	4	0	0	0	959.34	21.25%	-14.73%
J30	6	1	5	3	5.70	1.38%	81.16%

The LB found by this method is better in 62% of the instances and for other 10% it is the same found in the literature. Considering the set J30, the method closed 3 (50%) hard instances, found better LB for other 2 (33%) and reduced the average percentage GAP in more than 80%. For the class UBO, we found better LBs for 87% of the hard instances and reduced the percentage GAP in more that 10%. For the sets C and D, although the average GAP was not reduced, this method could find better LB for 4 (24%) hard instances and the same available in the literature for other 5 (29%) of set C.

The Table 2 shows the results obtained by our simplified method for the 58 hard instances and the 79 feasible instances of the set UBO500. This table follows the same structure of Table 1.

Table 2 – Results for simplified method

Group	Instances	Same LB	Better LB	Closed	Average Time (s)	GAP %	% GAP closed
UBO50	3	0	3	0	5.03	9.83%	66.99%
UBO100	10	0	8	0	211.58	17.00%	23.42%
UBO200	18	0	15	0	2,925.29	24.26%	9.12%
UBO500	79	42	26	0	25,388.69	9.21%	5.50%
C	17	3	2	0	119.98	17.03%	-10.31%
D	4	0	0	0	237.12	21.42%	-15.73%
J30	6	1	5	3	0.77	1.38%	81.16%

The LB found by this method is better in 43% of the instances and for other 34% it is the same available in the literature. This method could also prove optimality for 3 hard instances of set J30.Considering only the hard instances of class UBO, this method found better LB in 47% and reduced the GAP in more them 5%. For UBO500 instances, that Schutt et al. (2010) did not tested, our method could find the same LB in 53% of the instances and found better LBs in 33%. For the sets C and D, the GAP is in average more than 10% greater than the one in the literature.

Comparing the complete and the simplified methods, it is clear that the simplified one requires less computational effort, but the complete is important to reduce the GAP and to find better LBs for larger instances.

5. Conclusion

Considering the results obtained by the complete and the simplified methods presented in this paper, although the LP relaxation is important to reduce the average percentage GAP, it is clear that the complete method requires much more computational effort.

Comparing the results here obtained and the ones available at PSPLib (2012), we could reduce the percentage GAP for all but the sets C and D. The LBs were improved for 62% of the hard instances with the complete method and for 43% with the simplified one.

Comparing the method here proposed with the one presented by Schutt et al. (2010), their finite domain uses the Cumulative Scheduling Problem that is similar to the SRWP, but it does not allow preemption and the execution time is fixed. So their relaxation is stronger than the one used in this work. On the other hand, their Boolean satisfiability solving is used to add disjunction restriction to the model, each disjunction as a constraint. As the artificial resources introduced in this work are created considering more than one disjunction at a time, they are stronger as than the Boolean satisfability. Another important difference between the methods is that Schutt et al. (2010) method includes a branch-and-bound procedure and the one proposed here does not.

A weak point of our method is the fact that it needs a valid upper bound as input. Although the quality of the upper bound does not change the final LB, it may affect the constructive LB and the total execution time. Another important fact is that our method does not include a heuristic procedure, so if the known upper bound is not optimal then we will not find a better solution. On the other hand, the execution intervals obtained after the constructive part of our LB can be used in other exact methods or in heuristics.

6. References

Baptiste, P; Pape, C. L.; Nuijten, W. and Pape, C. (1999), Satisfiability Tests and TimeBound Adjustments for Cumulative Scheduling Problems, *Annals of Operations Research*, 305-333.

Bartusch, M.; Möhring, R. H. and Radermacher, F. J. (1988), Scheduling project networks with resource constraints and time windows, *Annals of Operations Research*, 199-240.

Blazewicz, J.; Lenstra, J. K. and Kan, A. H. G. (1983), Scheduling subject to resource constraints: classification and complexity, *Discrete Applied Mathematics*, 32, 11-24.

Brucker, P.; Drexl, A.; Mohring, R.; Neumann, K. and Pesch, E. (1999), Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research*, 112, 3-41.

Brucker, P. and Knust, S. (2000), A linear programming and constraint propagation-based lower bound for the RCPSP, *European Journal of Operational Research*, 127, 355-362.

Demassey, S.; Artigues, C. and Michelon, P. (2005), Constraint-Propagation-Based Cutting Planes: An Application to the Resource-Constrained Project Scheduling Problem, *Journal on Computing*, 17, 52-65.

Dorndorf, U.; Pesch, E. and Phan-Huy, T. (2000), A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints, *Management Science*, 46, 1365-1384.

Franck, B.; Neumann, K. and Schwindt, C. (2001), Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling, *OR Spectrum*, 23, 297-324.

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

Kooli, A.; Haouari, M.; Hidri, L. and Néron, E. (2010), IP-Based Energetic Reasoning for the Resource Constrained Project Scheduling Problem, *ISCO 2010 - International Symposium on Combinatorial Optimization*, 359-366.

Pinedo, M, *Planning and Scheduling in Manufacturing and Services*, Springer, New York, 2004.

Pisinger, D. (1997), A Minimal Algorithm for the 0-1 Knapsack Problem, *Operations Research*, 45, 758-767.

Schutt, A.; Feydy, T.; Stuckey, P. J. and Wallace, M. G., Solving the Resource Constrained Project Scheduling Problem with Generalized Precedences by Lazy Clause Generation, *Computer Science*, Cornell University, http://arxiv.org/abs/1009.0347v1, 2010.

PSPLIB., Project Scheduling Problem Library, http://129.187.106.231/psplib, 2012.