June 2009 *RR*-07.09

Lucio Bianco, Massimiliano Caramia

An Exact Algorithm to Minimize the Makespan in Project Scheduling with Scarce Resources and Feeding Precedence Relations

An Exact Algorithm to Minimize the Makespan in Project Scheduling with Scarce Resources and Feeding Precedence Relations

Lucio Bianco [*] Massimiliano Caramia [†]

Abstract

In this paper we study an extension of the Resource-Constrained Project Scheduling Problem (RCPSP) with minimum makespan objective by introducing as precedence constraints the so called "Feeding Precedences" (FP). For the RCPSP with FP we propose a new mathematical formulation and a branch and bound algorithm exploiting the latter formulation. The exact algorithm takes advantage also of a lower bound based on a Lagrangian relaxation of the same formulation. A computational experimentation on randomly generated instances and a comparison with the results achieved by a commercial solver, show that the proposed approach is able to behave satisfactorily.

Keywords: Branch and bound, Feeding precedences, Lagrangian relaxation

1 Introduction

In this paper we study an extension of the classical Resource-Constrained Project Scheduling Problem (RCPSP) with minimum makespan objective by introducing a further type of precedence constraints denoted as "Feeding Precedences" (FP), firstly introduced by Kis et al. (2004).

This problem happens in that production planning environment, like make-to-order manufacturing, which commonly requires the so-called project-oriented approach. In this approach a project consists of tasks each one representing a manufacturing process.

[*]Dipartimento di Ingegneria dell'Impresa, Università di Roma "Tor Vergata", Via del Politecnico, 1 - 00133 Roma, Italy. e-mail: bianco@disp.uniroma2.it

[†]Dipartimento di Ingegneria dell'Impresa, Università di Roma "Tor Vergata", Via del Politecnico, 1 - 00133 Roma, Italy. e-mail: caramia@disp.uniroma2.it

Due to the physical characteristics of these processes, the effort associated with a certain activity for its execution may vary over time. An example is that of the human resources that can be shared among a set of simultaneous activities in proportion variable over time. In this case the amount of work per time unit devoted to each activity and therefore its duration are not univocally defined.

This kind of problems is in general modelled by means of the so called Variable Intensity formulation, that is a variant of the Resource Constrained Project Scheduling Problem (see, e.g., Kis, 2005). As the durations of the activities cannot be taken into play, the traditional finish-to-start precedence relations (see, e.g., Kelley, 1963), so as the generalized precedence relations (see, e.g., Bartusch et al., 1988, and Elmaghraby and Kamburowski, 1992), cannot be used any longer, and there is the need for the feeding precedences (Kis et al., 2004, Kis, 2006).

Feeding precedences are of four types:

- *Start-to-%Completed (S%C) between two activities* (i, j). This constraint imposes that the processed percentage of activity j successor of i can be greater than $0 \leq g_{ij} \leq 1$ only if the execution of i has already started (see Figure 1).

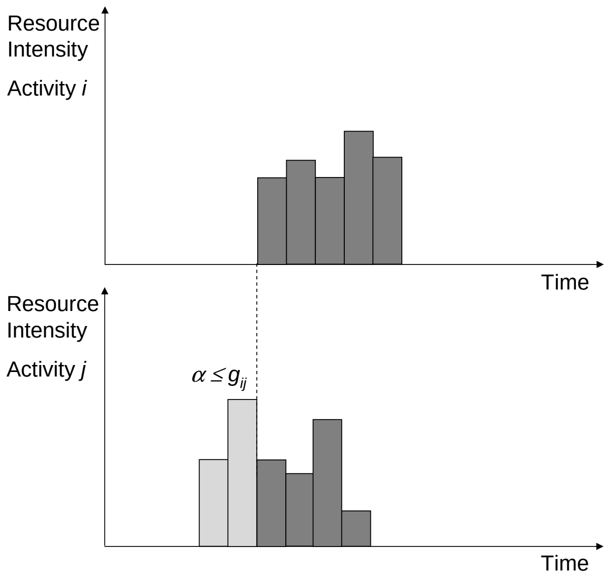

Figure 1: Example of a Start-to-%Completed constraint between activities i and j.

- *%Completed-to-Start (%CS) between two activities* (i, j). This constraint is used to impose that activity j successor of i can be executed only if i has been processed for at least a fractional amount $0 \leq q_{ij} \leq 1$ (see Figure 2).

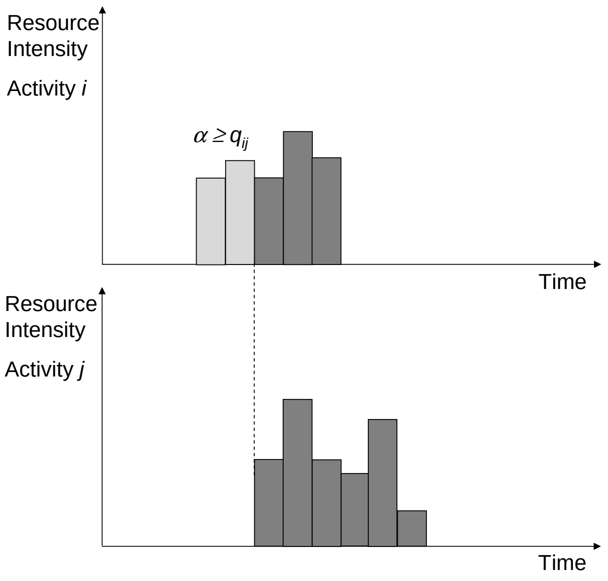

Figure 2: Example of a %Completed-to-Start constraint between activities i and j.

- *Finish-to-%Completed (F%C) constraints between two activities* (i, j). This constraint imposes that the processed fraction of activity j successor of i can be greater than $0 \leq g_{ij} \leq 1$ only if the execution of i has been completed (see Figure 3).

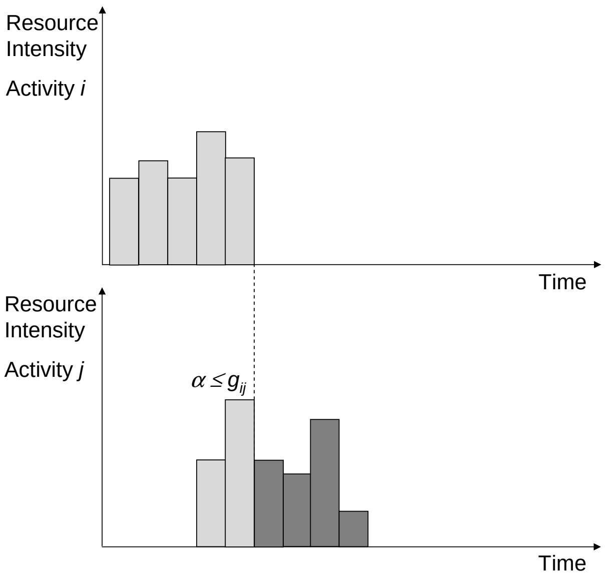

Figure 3: Example of a Finish-to-%Completed constraint between activities i and j.

- *%Completed-to-Finish (%CF) constraints between two activities* (i, j). This constraint imposes that the execution of activity j successor of i can be completed only if the fraction of i processed is at least $0 \leq q_{ij} \leq 1$ (see Figure 4).

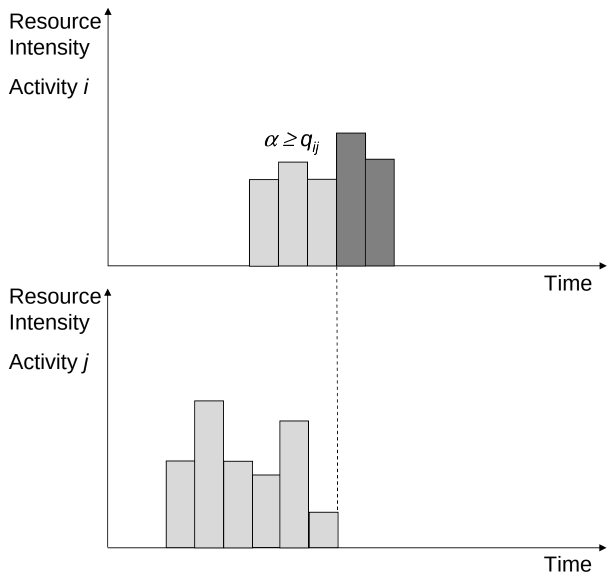

Figure 4: Example of a %Completed-to-Finish constraint between activities i and j.

In the following, we propose a new mathematical formulation of the RCPSP with FP constraints in terms of mixed integer programming (see Section 2). For this formulation a branch and bound algorithm has been designed (Section 3). The exact algorithm exploits a lower bound based on a Lagrangian relaxation of the mathematical formulation (see Bianco and Caramia, 2009). A computational experimentation on randomly generated instances and a comparison with a commercial solver are provided (Section 4).

2 The Mathematical Model

We will assume that the planning horizon within which all the activities have to be scheduled is $[0, T)$, where T is the project deadline, and it is discretized (without loss of generality) into T unit-width time periods $[0, 1), [1, 2), \dots, [T-1, T)$. Let us define with

- q^1_{ij}, the fraction of activity i that has to be at least completed in order to let activity j start;
- q^2_{ij}, the fraction of activity i that has to be at least completed in order to let activity j finish;
- g^1_{ij}, the fraction of j that can be at most completed before the starting time of activity i;

- g^2_{ij}, the fraction of j that can be at most completed before the finishing time of activity i;
- A, the set of activities to be carried out;
- A_1, A_2, A_3 and A_4, the sets of pairs of activities for which a $S\%C$, $\%CS$, $F\%C$, and $\%CF$ constraint exists, respectively;
- K, the set of renewable resources each one available in an amount of b_k units, with $k = 1, \dots, K$;
- q_{ik}, the amount of units of resource k necessary to carry out activity i.

Furthermore, let us consider the following decision variables

- x_{it}, the percentage of i executed till time period t.
- s_{it}, f_{it}, binary variables that assume value 1 if activity i has started or finished in a time period $\tau \leq t$, respectively, and assume value 0 otherwise.

Since the completion time of an activity $i \in A$ can be expressed as $f_i = \Big(T - \sum_{t=1}^{T} f_{it} + 1\Big)$, the objective function can be written as:

$$\min \{\max_{i \in A} f_i\} \quad = \quad \min \Big\{\max_{i \in A} \Big(T - \textstyle\sum_{t=1}^{T} f_{it} + 1\Big)\Big\}$$

The FP relations can be modelled as follows

$$\begin{array}{lll}
x_{jt} \leq s_{i,t-1} + g^1_{ij} & \forall (i,j) \in A_1, t = 1, \dots, T & (1) \\
s_{jt} \leq x_{i,t-1} + (1 - q^1_{ij}) & \forall (i,j) \in A_2, t = 1, \dots, T & (2) \\
x_{jt} \leq f_{i,t-1} + g^2_{ij} & \forall (i,j) \in A_3, t = 1, \dots, T & (3) \\
f_{jt} \leq x_{i,t-1} + (1 - q^2_{ij}) & \forall (i,j) \in A_4, t = 1, \dots, T & (4) \\
x_{it} \leq x_{i,t+1} & \forall i \in A, t = 1, \dots, T-1 & (5) \\
s_{it} \leq s_{i,t+1} & \forall i \in A, t = 1, \dots, T-1 & (6) \\
f_{it} \leq f_{i,t+1} & \forall i \in A, t = 1, \dots, T-1 & (7) \\
s_{iT} = f_{iT} = x_{iT} = 1 & \forall i \in A & (8) \\
s_{i0} = f_{i0} = x_{i0} = 0 & \forall i \in A & (9) \\
f_{it} \leq x_{it} \leq s_{it} & \forall i \in A, t = 1, \dots, T & (10) \\
\sum_{i=1}^{|A|} q_{ik}(x_{it} - x_{i,t-1}) \leq b_k & k = 1, \dots, K, t = 1, \dots, T & (11) \\
s_{it}, f_{it} \in \{0, 1\} & \forall i \in A, t = 1, \dots, T & (12) \\
x_{it} \geq 0 & \forall i \in A, t = 1, \dots, T & (13)
\end{array}$$

By posing $F = \max_{i \in A} \Big(T - \sum_{t=1}^{T} f_{it} + 1\Big)$, this problem can be rewritten as:

$$
\begin{array}{lll}
\min F & & \\
x_{jt} \leq s_{i,t-1} + g^1_{ij} & \forall (i,j) \in A_1, t = 1, \ldots, T & (1) \\
\ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots \\
x_{it} \geq 0 & \forall i \in A, t = 1, \ldots, T & (13) \\
F \geq \left(T - \sum_{t=1}^{T} f_{it} + 1\right) & \forall i \in A & (14)
\end{array}
$$

Constraints from (1) to (4) model $S\%C$, $\%CS$, $F\%C$ and $\%CF$ feeding constraints, respectively. Constraints (5) regulate the total amount processed of an activity $i \in A$ over time. Constraints (6) imply that if an activity $i \in A$ is started at time t, then variable $s_{i\tau} = 1$ for every $\tau \geq t$, and, on the contrary, if activity i is not started at time t, $s_{i\tau} = 0$ for every $\tau \leq t$. Constraints (7) are the same as constraints (6) when finishing times are concerned. Constraints (8) say that every activity $i \in A$ must start and finish within the planning horizon. Constraints (9) represent initialization conditions for variable s_{it}, f_{it}, x_{it} when $t = 0$. Constraints (10) force x_{it} to be zero if $s_{it} = 0$, and f_{it} to be zero if $x_{it} < 1$. Resource constraints are represented by relations (11). Constraints (12) and (13) limit the range of variability of the variables.

3 The Exact Algorithm Description

In the following, we describe the search tree structure, the Lagrangian lower bound, the branching rule and the node fathoming rule.

3.1 The search tree structure

The exact algorithm proposed exploits the mathematical formulation presented in the previous section and is based on branch and bound rules. The root node α_0 of the search tree is associated with the whole problem P_0 and two bounds, i.e., the trivial upper bound on the minimum makespan given by the time horizon T and the lower bound on the minimum makespan based on a Lagrangian relaxation of the resource constraints of the mathematical model (see the next section).

Each level of the search tree is associated with an activity in the set A of activities, which means that the tree has at most $|A|$ levels.

Assume that activity i is associated with the first level of the tree (see Figure 5); then level 1 is formed by T subproblems, denoted P_{1t} with $t = 1, \ldots, T$, each associated with a time slot t in the time horizon at which activity i can start at the very latest, i.e., at each node α_{1t} at level 1 of the tree the subproblem P_{1t} associated is obtained from P_0 (its parent) by imposing $s_{it} = 1$.

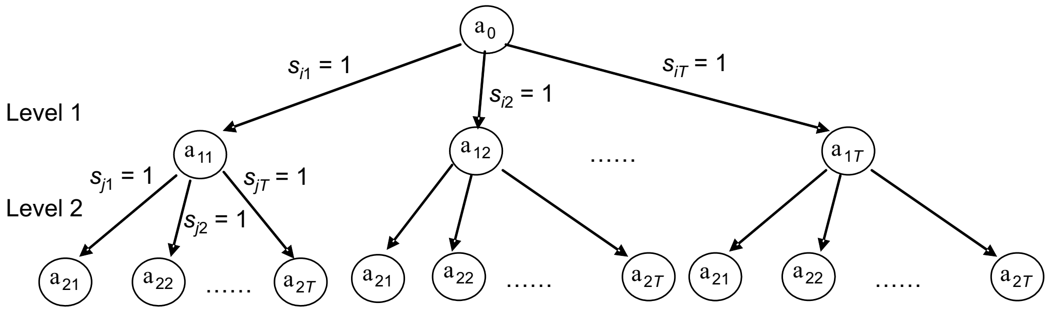

Figure 5: Two levels of the branch and bound tree. i and j are the activities associated to level 1 and level 2, respectively.

The same analysis described for level 1 can be applied to every level of the tree, i.e., from a subproblem $P_{i\tau}$ at level i we can generate T subproblems $P_{i+1,t}$ at level $i+1$, with $t = 1, \ldots, T$, each obtained from $P_{i\tau}$ by fixing $s_{jt} = 1$, where j is the activity associated with level $i+1$.

3.2 The Lagrangian lower bound

The lower bound used at each node of the search tree is based on a Lagrangian relaxation of the mixed integer formulation of the problem. The Lagrangian model P_{LaR} is obtained by penalizing the objective function with the resource constraints as follows:

$$\begin{array}{lll}
\min\left\{F \;-\; \sum_{k=1}^{K}\sum_{t=1}^{T} & \lambda_{kt}\left[b_k - \sum_{i=1}^{|A|} q_{ik}(x_{it} - x_{i,t-1})\right]\right\} & \\
x_{jt} \le s_{i,t-1} + g_{ij}^1 & \forall (i,j) \in A_1, t = 1, \ldots, T & (1) \\
\ldots & \ldots & \ldots \\
f_{it} \le x_{it} \le s_{it} & \forall i \in A, t = 1, \ldots, T & (10) \\
s_{it}, f_{it} \in \{0,1\} & \forall i \in A, t = 1, \ldots, T & (12) \\
x_{it} \ge 0 & \forall i \in A, t = 1, \ldots, T & (13) \\
F \ge \left(T - \sum_{t=1}^{T} f_{it} + 1\right) & \forall i \in A & (14)
\end{array}$$

where for each value of the λ_{kt} multipliers, the value of an optimal solution of P_{LaR} provides a lower bound to the optimal solution of problem P_0. The dual Lagrangian model is to find the λ_{kt} values maximizing the following problem

$$\begin{array}{lll}
\max_{\lambda_{kt}} \Big\{\min \Big\{F - \sum_{k=1}^{K} & \sum_{t=1}^{T} \lambda_{kt} \Big[b_k - \sum_{i=1}^{|A|} q_{ik}(x_{it} - x_{i,t-1})\Big]\Big\}\Big\} & \\
x_{jt} \leq s_{i,t-1} + g_{ij}^1 & \forall (i,j) \in A_1, t = 1, \dots, T & (1) \\
\dots & \dots & \dots \\
f_{it} \leq x_{it} \leq s_{it} & \forall i \in A, t = 1, \dots, T & (10) \\
s_{it}, f_{it} \in \{0,1\} & \forall i \in A, t = 1, \dots, T & (12) \\
x_{it} \geq 0 & \forall i \in A, t = 1, \dots, T & (13) \\
F \geq \Big(T - \sum_{t=1}^{T} f_{it} + 1\Big) & \forall i \in A & (14)
\end{array}$$

For this problem, in Bianco and Caramia (2009), a fast method to compute an estimate of the optimal λ_{kt}^* multipliers is proposed. It relies on a resolution of a linear program that guarantees the finiteness of the solutions to a problem which is a relaxation of the dual Lagrangian one. This avoids the use of more complex approaches to compute the λ_{kt}^* multipliers, like the subgradient optimization or other similar procedures. In our exact algorithm, we implemented this method since it requires very limited computing time, and offers better performance compared to other (non-Lagrangian) relaxations.

3.3 The branching and the node fathoming rules

We defined the branching rule by exploiting the characteristic of the feeding precedences by which if (i, j) is a pair of activities belonging to one of the four sets A_1, A_2, A_3, A_4, then activity i may constraint the starting time of activity j, that is the branching variable in our approach, whilst the opposite is not true. This means that, representing all the feeding precedences in a graph (see Figure 6) such that an arc (i, j) exists if there is a feeding precedence between i and j, the higher the number of successors of an activity, the higher should be the degree of constrainedness that this activity may impose to its successors. Therefore, the rule we adopt is that of assigning to each level the activity with the greatest number of successors (ties are broken arbitrarily). In the example in Figure 6, the levels of the search tree are associated e.g. with the following activities: 1, 3, 2, 4, 5, 6, 7, 8.

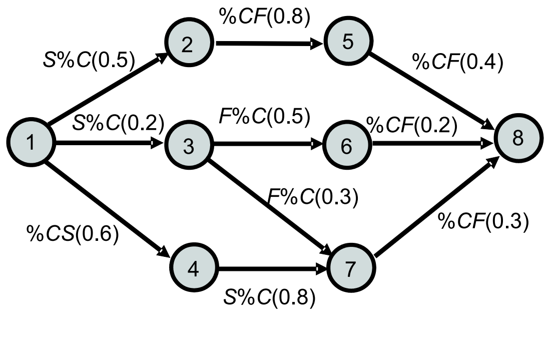

Figure 6: A graph representing the feeding precedences.

Let us now consider the bounding phase, with the consequent fathoming rules. Each subproblem P_{it} undergoes a bounding phase in which a lower bound on the minimum makespan is computed as done for the root node. Here we can have four alternative outcomes:

1. the time slot t of subproblem P_{it} is such that $t \geq UB^*$ where UB^* is the best upper bound found so far;
2. the mathematical program associated with the Lagrangian lower bound is empty, i.e., some feeding precedence relations cannot be obeyed;
3. the solution of the Lagrangian relaxation is not feasible with respect to some resource constraints;
4. the latter solution respects all the resource constraints.

Clearly, in the case 1, the tree is pruned and a backtracking phase is executed. Also in the case 2, the subtree generating from problem P_{it} is fathomed since a feasible solution cannot be found, with a consequent backtracking to the previous level; in the case 3, the Lagrangian relaxation solution value $La(P_{it})$, that is a lower bound for subproblem P_{it}, is compared to the best upper bound UB^* found so far; if $La(P_{it}) \geq UB^*$ then the tree is pruned again (with a consequent backtracking), otherwise the search is continued in a depth- (or breadth-) first search strategy.

In the last occurrence, i.e., in the case 4, the solution value $f(P_{it})$, obtained by substituting the x_{it}, s_{it}, f_{it} values obtained by the Lagrangian relaxation in P_{it}, is an upper bound for the latter subproblem and therefore it is an upper bound for the whole problem P_0. Now, if this solution to P_{it} satisfies the complementary slackness conditions, it is the optimal solution for P_{it} and the tree can be pruned, possibly updating UB^* to $f(P_{it})$ if the former is greater than the latter. If the solution is not optimal for P_{it}, then the search continues in a depth (or breadth) first search strategy possibly updating UB^* to $f(P_{it})$ if $UB^* > f(P_{it})$.

4 Computational Results

4.1 Implementation details

The implementation of our algorithm has been carried out in the C language. The performance of our approach has been compared to that of the commercial solver CPLEX, implementing the mathematical formulation presented in Section 2 in the AMPL language, version 8.0.0. The machine used for the experiments is a PC Core Duo with a 1.6 GHz Intel Centrino Processor and 1 GB RAM. Experiments have been generated with the following features:

- the number of activities $|A|$ has been chosen equal to 10, 20, 30, 40, 50, 60 and 70;
- a density fd of feeding precedences has been set equal to 30%;
- the number K of renewable resources has been kept equal to 4;
- an amount b_k of resource availability per period for each resource $k = 1, \dots, K$ has been set equal to 4;
- a request q_{ik} of resource $k = 1, \dots, K$ for every activity $i \in A$ has been assigned uniformly at random from 1 to 3;
- the values $g^1_{ij}, q^1_{ij}, g^2_{ij}, q^2_{ij}$ have been assigned uniformly at random in the range $(0.00, 1.00)$;
- the time horizon T, that is the starting upper bound at the root node of the search tree, has been fixed to 80.

4.2 Analysis of the results

Results, given as averages over five instances, are shown in Tables 1-6. In the following, we will denote our algorithm with `BB`. We listed the following values:

- `OPT_BB`, being the average values of the makespan, computed over the instances solved at the optimum, achieved by our algorithm; `#_Opt_BB`, being the number of instances, out of the five, solved at the optimum by our algorithm;
- `CPU_BB`, being the average CPU time (in seconds) elapsed by our algorithm to find the optimal solutions;
- the number `B&B` of nodes examined by our algorithm in the search tree;
- the average depth δ reached by our algorithm during the search in the branch and bound tree;
- $\rho_{\texttt{inf}}$, the average number of prunings related to infeasibility;
- $\rho_{\texttt{lb}}$, the average number of prunings related to the lower bounding procedure;

- $\rho_{\texttt{tot}}$, the total number of prunings executed by the branch and bound algorithm;
- $\phi_{\texttt{UB}}$, the first upper bound found by the branch and bound algorithm;
- CPU($\phi_{\texttt{UB}}$) the CPU time to find the first upper bound;
- $\sigma_{\texttt{UB}}$, the second upper bound found by the branch and bound algorithm;
- CPU($\sigma_{\texttt{UB}}$), the CPU time to find the second upper bound.

In Tables 1-4, we propose different implementations of BB. In particular, in Tables 1 and 2, to assess the role played by the Lagrangian relaxation based lower bound, we ran the algorithm with a depth-first-search strategy without (the first table) and with (the second table) the Lagrangian relaxation. In Table 1, we replaced the Lagrangian relaxation lower bound with a linear relaxation lower bound. As one can infer from the tables, the results clearly show that BB takes advantage of the Lagrangian relaxation in terms of both size of the instances solved and CPU time employed.

$\|A\|$	Opt_BB	#_Opt_BB	CPU_BB	B&B	$\rho_{\texttt{tot}}$	δ	$\phi_{\texttt{UB}}$	CPU($\phi_{\texttt{UB}}$)	$\sigma_{\texttt{UB}}$	CPU($\sigma_{\texttt{UB}}$)
10	5.2	5/5	0.5	578	85	7.8	8.4	0.2	7.8	0.3
20	9.0	5/5	55.6	578,253	195	17.6	16.8	10.4	16.4	12.2
30	14.5	2/5	1618.3	12,535,485	687	22.3	20.2	25.2	19.4	30.4
30	[13.7,14.8]	3/5	-	-	-	-	-	-	-	-
40	24.0	1/5	2912.3	100,254,128	912	33.0	31.0	81.4	30.0	124.0
40	[22.0,26.0]	4/5	-	-	-	-	-	-	-	-
50	-	0/5	-	-	-	-	-	-	-	-
50	[33.0,34.2]	5/5	-	-	-	-	-	-	-	-

Table 1: Performance of the branch and bound algorithm implemented with a depth-first-search strategy without the Lagrangian relaxation based lower bound.

Similarly, in Tables 3-4, we ran BB without (the first table) and with (the second table) the Lagrangian relaxation, using a breadth-first-search strategy. Also in this case the results highlight the advantages of using the latter lower bound. These experiments point out also another important aspect: indeed, by Tables 2 and 4, we notice that the results achieved by BB are comparable both in terms of size of solved instances and CPU times (there is a slight advantage for the depth-first-search looking at running times). This allows us to state that the proposed exact algorithm is quite robust.

We further note that when $|A| = 60$ and 70, BB (see Tables 2 and 4) for both the dept- and the breadth-first search strategy, is not able to solve all the five instances. Therefore,

$\lvert A\rvert$	Opt_BB	#_Opt_BB	CPU_BB	B&B	$\rho_{\texttt{inf}}$	$\rho_{\texttt{lb}}$	$\rho_{\texttt{tot}}$	δ	$\phi_{\texttt{UB}}$	CPU($\phi_{\texttt{UB}}$)	$\sigma_{\texttt{UB}}$	CPU($\sigma_{\texttt{UB}}$)
10	5.2	5/5	0.3	128	60	42	102	6.2	7.2	0.08	7.0	0.09
20	9.0	5/5	34.2	103,489	176	182	358	12.4	13.4	1.2	13.2	1.8
30	15.8	5/5	1248.9	2,512,829	371	654	1025	18.6	18.4	6.4	18.0	9.2
40	24.0	5/5	1923.4	18,128,442	715	948	1663	25.4	28.2	12.6	28.0	17.4
50	33.8	5/5	7024.5	78,202,790	894	1412	2306	31.2	40.4	18.2	40.0	25.2
60	48.0	1/5	7125.0	85,332,465	955	1525	2480	40.0	60.0	23.5	58.0	36.4
60	[45.0,48.5]	4/5	-	-	-	-	-	-	-	-	-	-
70	-	0/5	-	-	-	-	-	-	-	-	-	-
70	[64.4,68.6]	5/5	-	-	-	-	-	-	-	-	-	-

Table 2: Performance of the branch and bound algorithm implemented with a depth-first-search strategy with the Lagrangian relaxation based lower bound.

in these situations, we reported, just below the lines associated with $|A| = 60$ and $|A| = 70$, the (average) lower bound-upper bound gap when the algorithm terminates. It appears how this gap is very limited, i.e., less than 10%. The same situation is reported in Tables 1 and 3, for the lines associated with $|A| = 30, 40$ and 50.

$\lvert A\rvert$	Opt_BB	#_Opt_BB	CPU_BB	B&B	$\rho_{\texttt{tot}}$	δ	$\phi_{\texttt{UB}}$	CPU($\phi_{\texttt{UB}}$)	$\sigma_{\texttt{UB}}$	CPU($\sigma_{\texttt{UB}}$)
10	5.2	5/5	1.2	602	68	8.0	8.4	0.7	8.2	0.8
20	9.0	5/5	89.2	636,078	178	17.8	16.8	30.8	16.4	34.4
30	14.5	2/5	1725.4	15,042,582	589	23.6	20.2	227.2	19.8	285.6
30	[13.3,15.0]	3/5	-	-	-	-	-	-	-	-
40	24.0	1/5	3059.5	135,111,314	782	32.0	32.0	420.2	31.0	484.6
40	[21.5,26.5]	4/5	-	-	-	-	-	-	-	-
50	-	0/5	-	-	-	-	-	-	-	-
50	[32.6,34.6]	5/5	-	-	-	-	-	-	-	-

Table 3: Performance of the branch and bound algorithm implemented with a breadth-first-search strategy without the Lagrangian relaxation based lower bound.

4.3 Comparison with CPLEX

In this section, we show a comparison between BB and the commercial solver CPLEX (www.ilog.com). Note that a comparison with a state-of-the-art algorithm is not possible since, to the best our knowledge, there is no competing approach in this research area.

Table 5 shows the results achieved by CPLEX. We reported the following parameters:

$\|A\|$	Opt_BB	#_Opt_BB	CPU_BB	B&B	$\rho_{\texttt{inf}}$	$\rho_{\texttt{lb}}$	$\rho_{\texttt{tot}}$	δ	$\phi_{\texttt{UB}}$	CPU($\phi_{\texttt{UB}}$)	$\sigma_{\texttt{UB}}$	CPU($\sigma_{\texttt{UB}}$)
10	5.2	5/5	0.4	136	65	45	110	6.8	7.4	0.12	7.2	0.15
20	9.0	5/5	37.4	126,695	185	187	372	14.8	13.8	1.7	13.6	3.2
30	15.8	5/5	1325.6	2,763,570	405	673	1078	19.2	19.2	10.2	19.0	17.6
40	24.0	5/5	2089.5	25,466,299	702	822	1524	27.2	29.2	18.2	29.0	24.0
50	33.8	5/5	7142.2	85,748,882	821	1213	2034	33.8	43.6	23.4	41.8	35.6
60	48.0	1/5	7192.0	92,525,122	955	1410	2365	42.0	62.0	30.6	61.0	39.6
60	[44.5,48.5]	4/5	-	-	-	-	-	-	-	-	-	-
70	-	0/5	-	-	-	-	-	-	-	-	-	-
70	[62.4,69.2]	5/5	-	-	-	-	-	-	-	-	-	-

Table 4: Performance of the branch and bound algorithm implemented with a breadth-first-search strategy with the Lagrangian relaxation based lower bound.

$\|A\|$	Opt_CPLEX	#_Opt_CPLEX	CPU_CPLEX	B&B
10	5.2	5/5	0.7	653
20	9.0	5/5	67.6	789,243
30	14.5	2/5	1826.6	20,125,729
40	24.0	1/5	3185.2	125,258,240
50	-	0/5	-	-

Table 5: Performance of CPLEX.

- OPT_CPLEX, being the average values of the makespan, computed over the instances solved at the optimum, achieved by CPLEX;
- #_Opt_CPLEX, being the number of instances, out of the five, solved at the optimum by CPLEX;
- CPU_CPLEX, being the average CPU time (in seconds) elapsed by CPLEX to find the optimal solutions;
- the number B&B of nodes examined by CPLEX in the search tree.

The comparison between CPLEX and our algorithm points out how the latter was able to solve all the instances with fifty activities unlike the former that was able to solve all the instances only when $|A| = 20$. Indeed, CPLEX solved only two out of five instances when $|A| = 30$, one out of five instances when $|A| = 40$, and exceeded the time limit for every instances with fifty activities. Moreover, CPU_CPLEX is always greater than CPU_BB.

4.4 Analysis on the effectiveness of a heuristic upper-bound algorithm at the root

In Table 6, we report the effect on the performance of BB (implemented in depth-first-search strategy) of a possible effective upper-bound heuristic running at the root node. We simulated this by assuming an upper bound value at the root (see column UB_at_the_root) at distance two, one and zero from the optimal solution found by the exact algorithm, when this optimal solution has been found, and at distance two, one and zero from the best upper bound found by the exact algorithm, when the optimal solution was not found within the time limit.

$\|A\|$	UB_at_the_root	Opt_BB	CPU_BB	Time_to_Starting_UB
30	17.8	5/5	1154.2	10.4
	16.8	5/5	1127.4	52.3
	15.8	5/5	1025.8	112.4
40	26.0	5/5	1723.2	89.2
	25.0	5/5	1603.5	153.2
	24.0	5/5	1528.5	225.3
50	35.8	5/5	6003.9	145.3
	34.8	5/5	5834.1	257.6
	33.8	5/5	5676.9	301.5
60	50.2	2/5	6825.5	211.0
	49.2	2/5	6556.4	356.8
	48.2	3/5	6257.3	478.2
70	70.6	1/5	7112.2	355.2
	69.6	1/5	6987.4	498.4
	68.6	1/5	6759.7	577.7

Table 6: Performance of the branch and bound algorithm implemented with a depth-first-search strategy and different starting upper bounds.

The results show that the gain on the previously unsolved instances is modest: in particular when $|A| = 60$ either one or two more instances can be optimally solved, when we are at distance one or zero from the best upper bound found; when $|A| = 70$ this improvement is limited to one more instance, regardless of the distance to the best upper bound.

For the sake of completeness and in order to have a complete information to interpret the results, in column Time_to_Starting_UB we show the time (in seconds) needed to achieve the upper bound in column two by the exact algorithm implemented with the trivial upper bound at the root node. These results allow one to say that an effective upper-bound heuristic, in order to be competitive with our approach, should employ less CPU time than that reported in the last column of Table 6. Indeed, if it was not the case, it would be more profitable to use our algorithm to obtain the upper-bound value reported in such a column, and then restart the latter algorithm with this new value at the root ending up with an overall running time equal to the sum of the values reported in the forth and the fifth columns of Table 6.

Finally, we stress that, looking at the unsolved instances in Tables 2 and 4, it appears how the optimality gap is very limited, and therefore an effective heuristic has no chance to find strikingly different upper-bound values with respect to those achieved by `BB`. This confirms the good behavior of `BB` that is not deeply affected by the quality of the upper-bound value at the root of the search tree.

5 Conclusions

An exact algorithm for the project scheduling problem with feeding constraints, resource constraints and makespan minimization objective has been proposed. To the best of our knowledge, despite the practical importance of this problem, there is no exact algorithm tailored for such a problem in the literature. Computational experiments on randomly generated instances are promising, also in relation to a comparison with the commercial solver CPLEX. Indeed, with a time limit of two hours, our algorithm finds the minimum makespan on projects with 50 activities, while CPLEX is able to solve instances with up to 20 activities. Our algorithm was implemented with both the depth-first and the breadth-first search strategy producing comparable results, which highlights the robustness of the approach. We experimented our algorithm also by replacing the Lagrangian relaxation based lower bound with a simpler linear relaxation lower bound, showing how the branch and bound exploits the former lower bound to prune the search tree and to find good upper bounds quickly, without the need for an effective heuristic algorithm at the root. Future work will be devoted to design other branching rules as well as new lower bounds to improve the effectiveness of our approach.

References

[1] Bartusch, M., R.H. Mohring, F.J. Radermacher. 1988. Scheduling Project Networks with Resource Constraints and Time Windows, Annals of Operations Research **16**,

201-240.

[2] Bianco, L., M. Caramia. 2009. Minimizing the Completion Time of a Project Under Resource Constraints and Feeding Precedence Relations: a Lagrangian Relaxation Based Lower Bound, RR-03.09 - University of Rome "Tor Vergata", submitted.

[3] Elmaghraby, S.E.E., J. Kamburowski. 1992. The Analysis of Activity Networks under Generalized Precedence Relations (GPRs), Management Science **38(9)**, 1245–1263.

[4] Kelley, J.E. 1963. The critical path method: Resource planning and scheduling, Industrial Scheduling, J.F. Muth and G.L. Thompson (Editors), Prentice Hall, N.J., pp. 347–365.

[5] Kis, T., G. Erdõs, A. Márkus, J. Váncza. 2004. A Project-Oriented Decision Support System for Production Planning in Make-to-Order Manufacturing, ERCIM news **58**, 66–67.

[6] Kis, T. 2005. A branch-and-cut algorithm for scheduling of projects with variable-intensity activities, Mathematical Programming **103(3)**, 515–539.

[7] Kis, T. 2006. Rcps with variable intensity activities and feeding precedence constraints. In Perspectives in Modern Project Scheduling, Springer, pp. 105–129.