RAIRO-Oper. Res. 46 (2012) 373–409
DOI: 10.1051/ro/2012021

RAIRO Operations Research
www.rairo-ro.org

FLOW POLYHEDRA AND RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEMS

Alain Quilliot[1] and Hélène Toussaint

Abstract. This paper aims at describing the way Flow machinery may be used in order to deal with Resource Constrained Project Scheduling Problems (RCPSP). In order to do it, it first introduces the Timed Flow Polyhedron related to a RCPSP instance. Next it states several structural results related to connectivity and to cut management. It keeps on with a description of the way this framework gives rise to a generic Insertion operator, which enables programmers to design greedy and local search algorithms. It ends with numerical experiments.

Keywords. Scheduling with resource constraints, network flow theory.

Mathematics Subject Classification. 90-08.

1. Introduction

Dealing with *Resource Constrained Project Scheduling Problems* (RCPSP: see [10, 12, 20, 21, 41, 50]) means scheduling a set of tasks, which is submitted to temporal and cumulative resource constraints, in such a way that the induced *Makespan* value be the smallest possible. This problem, which can be viewed as an extension of the *Multiprocessor Scheduling* Problem (see [9, 39, 66, 69, 70]), is one of the problems which have been the most widely studied in Scheduling Theory: it may be related to many practical applications which involve industrial activity planning (see [12, 15, 42, 72]); at the same time, its theoretical analysis requires the use of sophisticated mathematical tools like linear programming, quadratic programming, and also of combinatorial tools like partially ordered sets

Received April 17, 2011. Accepted October 8, 2012.

[1] LIMOS, UMR CNRS 6158, Bat. ISIMA, Université Blaise Pascal,Campus des Cézeaux, BP 125, 63173 Aubiere, France. alain.quilliot@isima.fr

Article published by EDP Sciences

© EDP Sciences, ROADEF, SMAI 2012

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

and hypergraphs (see [28,29,34,36,38]). While the standard RCPSP problem only involve deterministic non pre-emptive tasks submitted to binary precedence relations and to restrictions on the use of a set of renewable resources, one may also address a very large number of possible extensions which involve pre-emption (see [29,60–62]), extended precedence relations (time lags: see [32]), non renewable resources (see [47,59]), non constant profiles, financial flows (see [47,72]), robustness in relation to uncertainty (see [24]), multiple task execution modes... Also, it is possible to handle this problem while focusing on other performance criteria than the makespan: criteria related to economical costs, to deadlines and penalties, to redundant resources... (see [1,22,43]). A recent survey about variants and extensions of the RCPSP is available in [40].

The RCPSP problems are difficult ones: whatever the way they are formulated (see [10,21,53]), they remain NP-Complete. Practically, getting exact results becomes hard as soon as the number of tasks exceeds 60 and that the number of resources is at least equal to 4 (see [31,51]). Generating benchmarks and characterizing their computational complexity may itself be viewed as a difficult problem (see [48,51]). When it comes to the design of exact methods, RCPSP problems are usually handled through Integer Linear Programming (see [28,47,56]), through a combination of branch and bound, cut generation and constraint propagation techniques (see [19,30,32,68]), or through extensions of multiprocessor scheduling algorithms (see [33,44,60]). Powerful lower bounds may be obtained through application of column generation techniques to specific linear programming models, through energetic reasoning processes, through pre-emption handling or through the computation of largest paths (see [11,18,23,29]). But efficient heuristics may also be designed: one may for instance refer to [24,62] for greedy algorithms based on priority rules, to [6,7] for very efficient algorithms based upon insertion techniques, to [5,16,35,51,52,65,78] for local search methods driven by metaheuristic scheme (Tabu, Simulated Annealing, memetic approaches), as well as to [46] for statistics about experiments. In case the problem is set according to a dynamic point of view, authors most often propose priority rule based algorithms, (see [7,62]).

Network Flow Theory (see [2,3,57]) is dedicated to the modelling and to the algorithmic handling of problems which involve the circulation of goods, people, money, energy or information. It has been essentially used in order to optimize the design of transportation and telecommunication systems (see [27]), or in order to help in managing the activity of gas or electricity distribution networks (see [2,58,67]). It proved itself to be a very powerful tool for the numerical handling of such problems, not only as a modelling tool, but also as a specific link between the linear programming machinery and purely combinatorial techniques. As a matter of fact, part of current research trends is about handling network flow problems while taking into account purely combinatorial constraints (see [25,67,79]).

The existence of a link between the RCPSP and Network Flow Theory has already be noticed by several authors (see [37,42,43,54,63]). Recently [8] studied the complexity of flow-based insertion problem for RCPSP with generalized prece-

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

dence relations. This link between the RCPSP and Network Flow Theory has been used in order to get ILP formulations of the RCPSP, as well as some specific heuristics: for example [37] proposed flow-based local search operators while [6, 7] proposed a flow-based insertion mechanism. Briand [17] proposed a schedule generation scheme using the operator insertion designed by Artigues. Still, few works have explicitly involved the network flow machinery in the design of algorithms. So, our goal is to make appear the way this link may help in designing fast and efficient generic methods, *i.e.*, methods which are going not to require too much implementation effort and which will easily reused in case of context changes. We are first going to recall and formalize the way RCPSP may be cast into the Network Flow framework, while introducing the *Flow Polyhedron Vertex Subset* related to a RCPSP instance. Next, we shall state several structural results about the connectivity of this vertex subset and about cut management. Finally, we shall derive from this theoretical work a generic *insertion* mechanism, close to the insertion mechanisms which were proposed in [6, 7], and which will be used in order to design greedy and local search randomized algorithms.

2. Network and Multi-Commodity Flow related to a RCPSP Instance

2.1. Preliminary notations and definitions

About sets, algorithms, lists, partial and linear orderings: we denote by $\leftarrow$ the value allocation operator: "$x \leftarrow \alpha$" means that the variable x takes the value α; so the symbol "=" is used inside algorithmic descriptions as a comparator or as a descriptor. We denote by Q the set of the rational numbers. If A is a set, we denote its cardinality by $\mathrm{Card}(A)$; if τ is some linear (or complete) order relation defined on some finite set X, then we consider τ as both a binary relation and a list; if A is some subset of X, we denote by $\mathrm{Min}(A, \tau)$ ($\mathrm{Max}(A, \tau)$) the smallest (largest) element of A according to τ; if x is some element in A, we denote by $\mathrm{Succ}(x, A, \tau)$ ($\mathrm{Pred}(x, A, \tau)$) the successor (predecessor) of x in A according to τ, which becomes undefined in case $x = \mathrm{Max}(A, \tau)$ ($\mathrm{Min}(A, \tau)$); if τ is some partial order relation, we denote by $\tau^{=}$ the relation (τ or =) and we denote by $Tr\ (\tau)$ the *transitive closure* of τ.

About graphs and networks (oriented graphs): we refer to the standard notation of C. BERGE (see [14]). An oriented graph (network) G with node (vertex) set Z and arc set E is denoted by $G = (Z, E)$. An arc e with origin node x and destination node y is denoted by (x, y). Such an oriented graph G is said to be *no circuit* if it does not contain any circuit. A partial graph of G is the restriction of G to some subset of the arc set E, while a subgraph of G is the restriction of G to some subset of the node set X.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

2.2. The Standard Non-Preemptive RCPSP Problem

An instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$ of the standard *Resource Constrained Project Scheduling Problem* (RCPSP) is defined by:

- a set V of non pre-emptive activities: every activity v in V is endowed with some *duration* $d_v > 0$ and must be run without any interruption during some time window with length d_v;
- a binary no circuit *precedence* relation $\ll$, which is defined on the set V: $v \ll w$ means that the activity v must be finished before the activity w starts;
- a finite renewable *resource* set K: the initial available amount of resource $k \in K$ is given by the component R_k of the *resource* vector $R = (R_k, k \in K)$; during the whole time it is run, activity v requires a $r_{k,v}$ amount of resource k to be available, and forbids any other activity to use this amount of resource k. Once it is over, activity v gives this resource back to the system.

Then solving such a RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$ means computing, for any activity $v \in V$, its starting time $T_v \geqslant 0$, in such a way that:

- if v and $w \in V$ are such that $v \ll w$, then $T_v + d_v \leqslant T_w$; (*Precedence Constraint*);
- at any instant $t \geqslant 0$, and for any resource $k \in K$, $\sum_{v \in U(T,t)} r_{k,v} \leqslant R_k$, where $U(T,t) = \{v \in V \text{ such that } T_v \leqslant t < T_v + d_v \} \subseteq V$ is the set of the activities which are run at time t; (*Resource Constraint)*;
- the *makespan Makespan* $(T) = \mathrm{Sup}_{v \in V} (T_v + d_v)$ is the smallest possible.

Any V-indexed *time vector* T which is feasible for the above *Resource* and *Precedence* constraints is called a feasible RCPSP *schedule* for the activity set V.

2.3. Linking Network Flows with RCPSP: Timed Flows

Recall: *Network Flows and Multi-commodity Flows*
Given a network $G = (Z, E)$, *i.e.* an oriented graph with node (vertex) set Z and arc set E, together with a Q-valued function ϕ defined on the node set Z, we say that a Q-valued E-indexed vector f is a *ϕ-flow vector* iff:

$$\forall z \in Z, \quad \sum_{z \text{ is the origin of } e} f_e = \sum_{z \text{ is the destination of } e} f_e = \phi(z) \quad (\textit{Extended Kirshoff Law})$$

If I is some *commodity* set, if $\phi = (\phi(i),\ i \in I)$ is a *commodity function*, *i.e.*, if every $\phi(i)$, $i \in I$, is a Q-valued function defined on the node set Z, then we call *ϕ-multi-commodity flow vector* any collection $f = (f(i),\ i \in I)$, where every $f(i)$, $i \in I$, is a *$\phi(i)$-flow vector*.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

The Activity Network related to a RCPSP instance

Let us consider now a RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$. We associate with $\boldsymbol{I}$ the *Activity Network* $\boldsymbol{N}(V) = (V^*, E^*)$ by introducing two auxiliary nodes *Start* and *End*, and by setting:

- $V^* = V \cup \{Start, End\}$= node set of $\boldsymbol{N}(V)$;
- $E^* = \{(v, v'), v, v' \in V\} \cup \{(Start, v), (v, End), v \in V\} \cup \{(End, Start), (Start, End)\}$ = arc set of $\boldsymbol{N}(V)$.

We provide the arc set E^* of the network $\boldsymbol{N}(V)$ with a E^*-indexed *length vector* d^*, by setting:

- for any $v \in V$, $d^*_{(\mathrm{Start,v})} = 0$;
- $d^*_{(\mathrm{End,Start})} = -\infty$;
- for any $v \in V$, $w \in V \cup \{End\}$, $d^*_{\mathrm{v,w}} = d_v$.

We also define on the node set V^* of $\boldsymbol{N}(V)$ a *commodity vector* r^*, by setting for every resource $k \in K$, and for any v in V^*: if $v \in V$ then $r^*_{k,v} = r_{k,v}$ else $r^*_{k,v} = R_k$.

We define the *precedence arc subset* $E^*_{\ll}$ by setting: $E^*_{\ll} = \{(v, v'), v, v' \in V$ such that $v\ Tr\ (\ll) v'\} \cup \{(Start, \mathrm{v}), (v, End), v \in V\}$, where $Tr\ (\ll)$ denotes the *transitive closure* of the $\ll$ relation.

Associating a r^*-multi-commodity flow vector with a feasible solution T of the RCPSP instance I

Let us suppose now that T is some feasible solution of this RCPSP instance, and let us denote by Δ the *Makespan* value of T. Following [6,7], using for instance a geometrical representation of the schedule T (Gantt Chart) and proceeding by induction on the starting time $T(v)$, $v \in V$, one may easily derive from T a multi-commodity flow vector $F = (F(k), k \in K)$, which is defined on the *Activity Network* $\boldsymbol{N}(V)$, and which is such that, for any k in K:

- for any activity v in V, $\sum_{v \text{ is the origin of } e} F(k)_e = \sum_{v \text{ is the destination of } e} F(k)_e = r_{k,v}$;
- $\sum_{Start \text{ is the origin of } e} F(k)_e = \sum_{Start \text{ is the destination of } e} F(k)_e = R_k$;
- $\sum_{End \text{ is the origin of } e} F(k)_e = \sum_{End \text{ is the destination of } e} F(k)_e = R_k$.

It comes that $F = (F(k), k \in K)$ is a r^*-multi-commodity flow, which may be viewed as transporting the resources $k \in K$, from the source *Start* to the end-node *End*, while providing the activities $v \in V$ with the resources they require. We may identify the *support arc subset* $E(F, \ll)$ of F by setting: $E(F, \ll) = E^*_{\ll} \cup \{(v, w), v, w \in V$ such that $F_{(\mathrm{v,w})}$ is non null$\}$. Clearly, $E(F, \ll)$ is *no circuit*.

Figure 1 shows an example of a Gantt chart which gives rise to an ad hoc flow representation related to the instance described in Table 1. The Gantt chart represents any activity x as a rectangle of length (respectively height) equal to the duration (respectively resource consumption) of x. For example activity 3 has a duration $d_3 = 1$ and needs 2 units of resource. The flow representation underlines the exchange of resources between activities.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

TABLE 1. Example of RCPSP instance with 5 activities and 1 resource of capacity 4.

Activities	Duration	Resource requirement	Predecessors
1	2	2	–
2	1	1	1
3	1	2	–
4	3	1	–
5	2	2	3

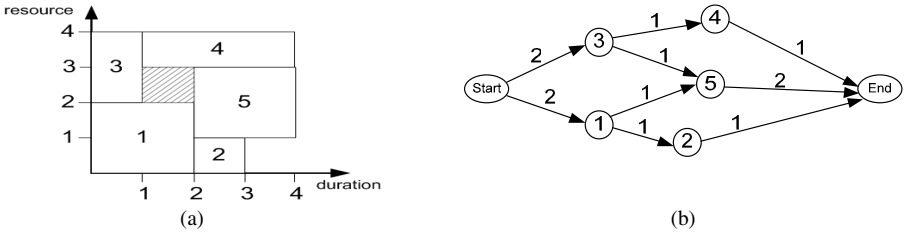


FIGURE 1. (a) Gantt chart related to instance of Table 1 – (b) Flow representation (the costs on the edges are the flow values).

Timed Flows

Also, if we extend the time vector T to $V \cup \{Start, End\}$ by setting:

- $T_{Start} = 0$;
- $T_{End} = \Delta = Makespan\ (T) = \mathrm{Sup}_{v \in V}\ (T_v + d_v)$;

then, for any arc $e = (v, w)$ in the arc set E^* of $\boldsymbol{N}(V)$, the following implication becomes true:

$$e = (v, v') \in E(F, \ll) \Rightarrow (T_w \geq T_v + d_v \Leftrightarrow T_w \geq T_v + d^*_e) \qquad \text{(P1)}$$

Conversely, let us call *no circuit r^*-flow* vector any r^*-multi-commodity flow vector F such that $E(F, \ll)$ is no circuit. Then we define a *timed (r^*, d^*)-flow* as being any pair (F, T) made of a no circuit r^*-multi-commodity-flow vector F and a time vector T such that (P1) is true. One easily checks that if (F, T) is such a timed (r^*, d^*)-flow defined on $\boldsymbol{N}(V)$, then the restriction of T to V is a feasible schedule for $\boldsymbol{I} = (V, K, R, r, d, \ll)$. This notion of *Timed Flow* provides us with a synthetic framework for the simultaneous handling of routing and scheduling problems. It allows us to reformulate the RCPSP Problem as follows:

Reformulation Scheme

Solving the RCPSP instance $\boldsymbol{I}$= (V, K, R, r, d, ≪) means computing, on the Activity network $\boldsymbol{N}(V)$, provided with the length vector d^*, with the commodity vector r^* and with the precedence arc subset $E^*_{\ll}$, a timed (r^*, d^*)-flow (F, T) such that T_{End} is the smallest possible.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

The Flow framework provides programmers with a tool for the design of generic algorithmic schemes, which is, sometimes, better-fitted than the very general ILP framework. The above Reformulation scheme open the way to the use of the flow machinery in the design of RCPSP algorithms.

2.4. A Connectivity Theorem

The aim of this section is to set a theoretical basis for the management of RCPSP problems through the use of Flow algorithms: though flow RCPSP representations were already used in the past, it was always according to an empirical approach. More specifically, part of the efficiency of the algorithmic network flow machinery derives from the structural properties of the network flow polyhedron: as a matter of fact, most flow algorithms may be viewed as primal-dual versions of the Simplex algorithm, and running those algorithms may be viewed as performing some walk on the vertex set of this network flow polyhedron. Thus, one may ask in a natural way about the structural properties of the subset of the network flow polyhedron node set which is defined by no circuit r^*-flow vectors.

The No Circuit r^*-Flow Polyhedral Vertex Set related to the RCPSP instance I

The set of all r^*-multi-commodity flow vectors $F \geqslant 0$ defined on the *Activity Network* $\boldsymbol{N}(V)$ defines a bounded polyhedron, which we denote by P_{r*}. It is known that such a r^*-multi-commodity flow vector F is a *vertex* of P_{r*} if and only if it contains no *non null alternated cycle*, that means no cycle $(v_0, v_1, \ldots, v_{\mathrm{n}} = v_0)$ such that:

- n is even and all the nodes $v_0, v_1, \ldots, v_{\mathrm{n}-1}$ are distinct (the cycle is elementary);
- there exists $k \in K$ such that:
  - the arcs $(v_0, v_1), (v_2, v_3), \ldots, (v_{\mathrm{n}-2}, v_{\mathrm{n}-1})$ are all endowed with non null $F(k)$ values;
  - the arcs $(v_2, v_1), (v_4, v_3), \ldots, (v_0, v_{\mathrm{n}-1})$ are all endowed with non null $F(k)$ values.

We denote by $S_{\mathrm{r}*}$ the vertex set of this polyhedron. It is known that $S_{\mathrm{r}*}$ is endowed with a canonical *adjacency* relation R, which corresponds to the moves which are performed by the *Simplex* Algorithm when running a linear program on a constraint set defined by $S_{\mathrm{r}*}$. This adjacency relation R may be characterized as follows:

- let Γ be some even elementary cycle $(v_0, v_1, \ldots, v_n = v_0)$ in $\boldsymbol{N}(V)$: we define the *alternated cycle flow* related to Γ as the flow vector f^{Γ} which is defined by:
  - $f_e^{\Gamma} = +1$ for any arc $e = (v_0, v_1), (v_2, v_3), \ldots, (v_{\mathrm{n}-2}, v_{\mathrm{n}-1})$;
  - $f_e^{\Gamma} = -1$ for any arc $e = (v_2, v_1), (v_4, v_3), \ldots, (v_0, v_{\mathrm{n}-1})$.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

- F, F' in S_{r*} are R-adjacent if there exists some resource $k_0 \in K$, some even elementary cycle Γ and some number $\lambda \geqslant 0$, such that we have:
  - for any $k \neq k_0$, $F(k) - F'(k) = 0$;
  - $F'(k_0) - F(k_0) = \lambda.\ f^{\Gamma}$.

In such a case, the value λ is unique, and we also say that F' derives from F through *redirection of F(k) on the cycle* Γ.

It comes from Linear Programming Theory that the vertex set S_{r*} of the polyhedron P_{r*} is connected for this non oriented adjacency relation R. *Redirection* processes and search into the vertex set S_{r*} are at the core of the classical Network Flow algorithmic framework. Clearly, casting RCPSP into this framework mean that we intend to perform thoses processes. Since we are required to deal with no circuit r^*-multi-commodity flows, we are led in a natural way to investigate whether restricting the *Redirection* scheme to the specific vertices of P_{r*}which define no circuit r^*-multi-commodity flow vectors maintains this connectivity property. In order to do it, we denote by SN_{r*} the restriction of S_{r*} to the no circuit r^*-multi-commodity flow vectors, that means to the r^*-multi-commodity flow vectors F such that the *support arc subset* $E(F, \ll)$ is *no circuit*. We call this set SN_{r*} the *No Circuit* r^* *-Flow Polyhedral Vertex Set.* Then we may state:

Theorem 2.1 (**Connectivity Theorem).**

If we suppose that, for any $k \in K$, v, $w \in V$, *we have:* $r_{k,v} + r_{k,w} \leqslant R_k$ *(Parallelism Hypothesis), then the No Circuitr***-Flow Polyhedral Vertex Set* SN_{r*} *is connected for the canonical adjacency relation R.*

Comment: of course, the meaning of this statement is that one may think into handling timed flows (and consequently RCPSP instances) while using the classical local search procedures (cancelling cycle procedures...) which are part of the Network Flow Theory machinery.

Proof.

Let us first define a *linear* r^*-multi-commodity-flow vector as being a no circuit r^*-multi-commodity-flow vector $F \geqslant 0$ which is such that the transitive extension of the support arc set $E(F,\ll)$ is linear. So we denote by SNL_{r*} the subset of SN_{r*} which is made with linear r^*-multi-commodity-flow vectors. If σ is some linear ordering of $V \cup \{Start, End\}$, which is compatible with $\ll$, we denote by $SN_{r*}(\sigma)$ the subset of SN_{r*} which corresponds to the case when σ may be viewed as a linear extension of the transitive extension of $E(F, \ll)$. We first state:

Lemma 2.2 (SN_{r*} **(σ) is connected for the** R **relation).**

Proof-Lemma. If F *is in* $SN_{r*}(\sigma)$, *then* $E(F,\ll)$ *is included into the arc set* $A(\sigma) = \{(v, w),\ v \in V \cup \{Start\},\ w \in V \cup \{End\},$ *such that* $(v\,\sigma\,w)\}$. *Conversely any multi-commodity flow vector* $F \geqslant 0$ *which is such that* $E(F, \ll) \subseteq A(\sigma)$, *and which does not admit any alternated cycle, is in* $SN_{r*}(\sigma)$. *But* $SN_{r*}(\sigma)$ *is nothing more than the vertex set of the polyhedron which is defined by imposing* F *to be null on the arcs which are not in* $A(\sigma) \cup \{(End, Start)\}$, *while the relation* R *is the adjacency relation related to this polyhedron. We deduce the result from the fact*

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

that any polyhedron vertex set is connected for its canonical adjacency relation. End-Lemma.

Lemma 2.3. *Let us suppose that, for any* $v \in V$, $k \in K$, $r_{k,v} \neq 0$ *and that the parallelism condition: for any v, w in V, we have:* $r_{k,v} + r_{k,w} \leqslant R_k$, *holds. Then:*

- *If F and F' are both in the set* SNL_{r*}, *then there exists a R-path from F to F';* (P2)
- *for any linear ordering* σ *of* $V \cup \{Start, End\}$, *which is compatible with* $\ll$, *the intersection of* SNL_{r*} *and* $SN_{r*}(\sigma)$ *is non empty.* (P3).

Proof-Lemma. (P3) derives in a trivial way from the fact that, for any $v \in V$, $k \in K$, $r_{k,v} \neq 0$.

In order to prove (P2), let us consider two *linear* r^*-multi-commodity-flow vectors F and F' such that F is in $SN_{r*}(\sigma)$ and F' is in $SN_{r*}(\tau)$, with $\sigma \neq \tau$. Because of (P3) and Lemma 1, we may choose F in such a way that for any $k \in K$, and for any v, $w \in V \cup \{Start, End\}$, such that w is the successor of v according to σ, v sends a non null $F(k)$ value to w (it is easy to check the existence of an element of $SN_{r*}(\sigma)$ which satisfies this property). Since $\sigma \neq \tau$, there must exist v, w, consecutive according to σ, such that $w \tau v$, and we may choose v in such a way that it is the smallest possible with this property according to σ. Let us denote by u and t respectively the predecessor of v and the successor of w according to σ. For any k, we have $F(k)_{(u,v)} \neq 0$ and $F(k)_{(w,t)} \neq 0$.

Let us consider now some resource k, together with the following subsets X and Y of $V \cup \{Start, End\}$, and the following flow amounts Q_1, Q_2, Q_3, Q_4:

- $X = \{x \in V \cup \{Start, End\}$ such that $x \sigma v\}$;
- $Y = \{y$ such that $w \sigma y\}$;
- $Q_1 =$ flow $F(k)$ amount which starts from X and which arrives into w;
- $Q_2 =$ flow $F(k)$ amount which starts from v and which arrives into Y;
- $Q_3 =$ flow $F(k)$ amount which starts from X and which arrives into Y;
- $Q_4 = F(k)_{(v,w)}$.

We see that: (P4)

- $R_k = Q_1 + Q_2 + Q_3 + Q_4$;
- $r_{k,v} = Q_2 + Q_4$;
- $r_{k,w} = Q_1 + Q_4$.

By combining those equalities (P4) with the inequality $r_{k,v} + r_{k,w} \leqslant R_k$ and with the relation $F(k)_{(v,w)} \neq 0$, we get that Q_3 must be non null, which means that there must exist x in X and y in Y such that $F(k)_{(x,y)} \neq 0$. So we may redirect $F(k)$ on the cycle $\Gamma = (x, w, v, y, x)$. By repeating this process as long as $F_{(v,w)} \neq 0$ we can make in such a way that $F_{(v,w)} = 0$.

Once it has been done, we still have $F(k)_{(u,v)} \neq 0$ and $F(k)_{(w,t)} \neq 0$, for any k in K. According to the previous process, there must exist, for any k in K, a node x in X and a node y in Y such that $F(k)_{(x,w)} \neq 0$ and $F(k)_{(v,y)} \neq 0$. In

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

case $x \neq u$, we redirect $F(k)$ on the cycle $\Gamma = (u, w, x, v, u)$. By the same way, in case $y \neq w$, we redirect $F(k)$ on the cycle $\Gamma = (v, t, w, y, v)$. Then we get that $F(k)_{(u,w)} \neq 0$ and $F(k)_{(v,t)} \neq 0$. We end the process by considering x' which provides a $F(k)$ flow amount to v, y' which receives $F(k)$ flow amount from w, and by redirecting $F(k)$ on the cycle $\Gamma = (w, v, x', y', w)$. At this time, F becomes a linear element of the set $SN_{r*}(\sigma')$, related to the linear ordering σ' which derives from σ by permuting v and w. We easily conclude by induction on the number of permutations which make possible turning σ into τ. *End-Lemma.*

Clearly, combining both previous lemmas allows us to conclude to the R-connectivity of SN_{r*} in the case when, for every activity v in V and every resource k in K, the quantity $r_{k,v}$ is non null.

In order to get our result in the general case, we use a trick which involves topology. Let $\delta > 0$ be a small positive number. For every activity v and any resource k, such that $r_{k,v} = 0$, we replace $r_{k,v}$ by δ, and R_k by $R_k + \text{Card}(V(k)).\delta$, where $V(k) = \{v \in V \text{ such that } r_{k,v} = 0\}$. We denote by $S^{\delta}_{\text{r}*}$ and SN^{δ}_{r*} the respective related polyhedron vertex sets and by R^{δ} the related adjacency relation. It comes from above that SN^{δ}_{r*} is connected for the relation R^{δ}. Also, we see that if F is some vertex in $SN_{\text{r}*}$, then the r*-multi-commodity-flow vector F^{δ} defined by:

- for any v and any k such that $r_{k,v} = 0$, $F^{\delta}(k)_{(Start,v)} = \delta = F^{\delta}(k)_{(v,End)}$;
- $F^{\delta}(k)_{(Start,s)} = \text{Card}(V(k)).\delta$;
- for any other pair (e, k), k in K, e in the arc set of the network $\boldsymbol{N}(V)$, $F^{\delta}(k)_e = F(k)_e$;

is no circuit and does not admit any non null alternated cycle, and thus is in $SN^{\delta}_{\text{r}*}$.

Let us now consider two r^*-multi-commodity-flow vectors F and H in $SN_{\text{r}*}$. F^{δ} and H^{δ} can be connected by a path $\gamma^{\delta} = (F^{\delta}_0, F^{\delta}_1, F^{\delta}_n = H^{\delta})$ (where each F^{δ}_{i} is a flow vector) in $SN^{\delta}_{\text{r}*}$, which means that it is possible to find coefficients $\lambda^{\delta}_0, \ldots, \lambda^{\delta}_{\text{n}-1}$, and cycles $\Gamma^{\delta}_0, \ldots, \Gamma^{\delta}_{\text{n}-1}$, in such a way that, for any $i = 0 \ldots n$-1: $F^{\delta}_{\text{i}+1} = F^{\delta}_{\text{i}} + \lambda^{\delta}_{\text{i}}.f^{\Gamma\delta,\text{i}}$, where $f^{\Gamma\delta,\text{I}}$ is the alternated cycle flow which derives from $\Gamma^{\delta}_{\text{i}}$. Since V is finite, a compacity argument allows us to suppose that:

- for any $i = 0 \ldots n$-1, $\Gamma^{\delta}_{\text{i}} = \Gamma_{\text{i}}$ does not depend on δ;
- $\lambda^{\delta\text{i}}$ converges to some coefficient λ_{i} when δ converges to 0.

Then it becomes easy to deduce a sequence $\gamma = (F_0, F_1, F_{\text{n}} = H)$ by setting, for any $i = 0 \ldots n$: $F_{i+1} = F_i + \lambda_i.\Gamma_i$. Every F_i defined this way is clearly no circuit. Also, it does not contain any non null alternated cycle. We deduce that γ defines a R path from F to H in $SN_{\text{r}*}$. *End-Theorem.*

Remark 2.4. the *Parallelism* hypothesis cannot be removed. If we consider for example the following RCPSP instance:

- $V = \{A, B, C, D\}$; $A \ll B$; $C \ll D$; $A \ll D$;
- $K = \{\alpha\}$; any activity in V has duration equal to 1; the total amount of resource α is 4;

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

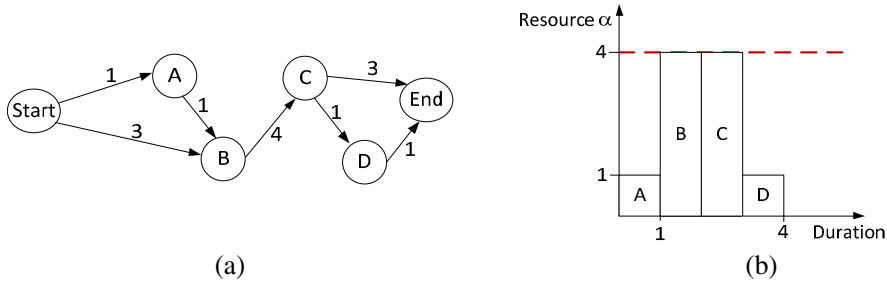


FIGURE 2. (a) Timed flow F – (b) Gantt chart which represents the timed flows F.

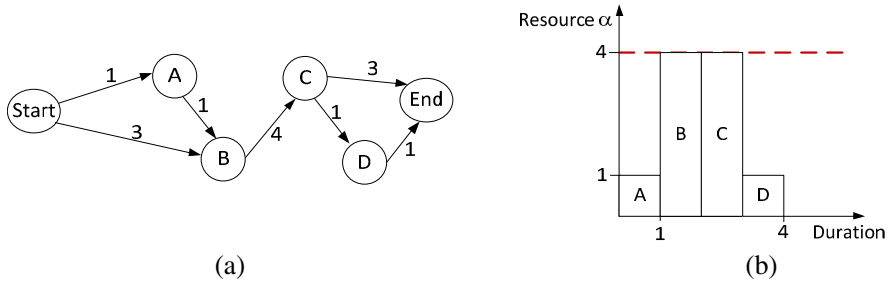


FIGURE 3. (a) Timed flow H – (b) Gantt chart which represents the timed flows H.

– A and D require 1 unit of the resource α; C and B require 4 units of the resource α;

then we notice that there exist only two related timed flows F and H, respectively defined by:

– $F_{(Start,\mathrm{A})} = 1$; $F_{(Start,\mathrm{B})} = 3$; $F_{(\mathrm{A},\mathrm{B})} = 1$; $F_{(\mathrm{C},\mathrm{D})} = 1$; $F_{(\mathrm{B},\mathrm{C})} = 4$; $F_{(\mathrm{C},End)} = 3$; $F_{(\mathrm{D},End)} = 1$;
– $H_{(Start,\mathrm{A})} = 1$; $H_{(Start,\mathrm{C})} = 3$; $H_{(A,C)} = 1$; $H_{(\mathrm{B},\mathrm{D})} = 1$; $H_{(\mathrm{C},\mathrm{B})} = 4$; $H_{(\mathrm{B},End)} = 3$; $H_{(\mathrm{D},End)} = 1$.

But we also see that the difference of those two timed flows is not the multiple of any alternated cycle flow. Figure 2 illustrates the timed flow F while 3 illustrates timed flow H.

3. The Match-Flow and Insertion-Flow Problems

As we told it inside the previous section, the main motivation for introducing the *Timed Flow* formalism is provided by the prospect of applying ad hoc network

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

flow algorithmic tools to RCPSP instances. So the current Section 3 is going to be devoted to the description of those basic algorithmic components which will be used in Section 4 in order to derive flow algorithms for RCPSP.

The algorithms which will be described in Section 4 works while performing *insertion/removal* processes which may be compared with those which have been proposed in [6, 7]: the basic difference lays upon the fact that every time the insertion/removal of some activity is performed, it involves the resolution of a specific *Insertion Flow* sub-problem related to a *Cut* of the currently inserted activity set; so, the related resolution process updates all the flow values which express the flow transportation between both sides of this *Cut*, and may be viewed as an implementation of the *Connectivity Theorem* of Section 2. More precisely, at any time during the process of some RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$, we are provided with some *Inserted Activity* subset W of V, with a no circuit r^*-multi-commodity flow vector F defined on the *Activity Network* $\boldsymbol{N}(W)$, and with two positive (or null) Q-valued time vectors T and T^*, both with indexation on W^*, in a such a way that, for any v in W: (P5)

- T_v = Length of a largest path from *Start* to v in the *Support Partial Activity Network* defined by $E(F,\ll)$, for the length vector d^*;
- T^*_v = Length of a largest path from v to *End* in the *Support Partial Activity Network* defined by $E(F, \ll)$, for the length vector d^*.

Clearly, this pair (F, T) defines a timed (r^*, d^*)-flow. Then, performing an *Insertion* means picking up some activity v_0 which is not in W, and turning (F, T), through some local computation process, into a convenient timed (r^*, d^*)-flow defined on the *Activity Network* $\boldsymbol{N}(W \cup \{v_0\})$. The related insertion mechanism involves a *Cut*, *i.e.* a partition of W into two subsets U and $W - U$, such that no flow amount goes from to $(W - U) \cup \{End\}$ to $U \cup \{Start\}$: therefore, the insertion process makes v_0 receive flow values from $U \cup \{Start\}$ and give them back to $(W - U) \cup \{End\}$. Performing a *Removal* means reversing this operation. In order to explain those mechanisms in an accurate way, we shall introduce (next subsections) the notions of *Match Flow* and *Insertion Flow*. Meanwhile, we may illustrate the general insertion/removal mechanism on the instance described in Table 1 through the drawings of Figure 4 which represents a partial solution with 4 activities and a cut ($U = \{1, 3\}$, $W - U = \{2, 4\}$): the aim is to insert activity $v_0 = 5$ (with $d_5 = 2$ and $r_5 = 2$) into this cut. The resultant flow is given in Figure 5.

3.1. The Match-Flow Problem

Match Flows

Let (X, E) be some bipartite graph, $X = A \cup B$ be a partition of X into two disjoint independent sets, Π, Π^* be two positive (or null) Q-valued functions, with respective domains A and B. We also suppose that we are provided with two

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

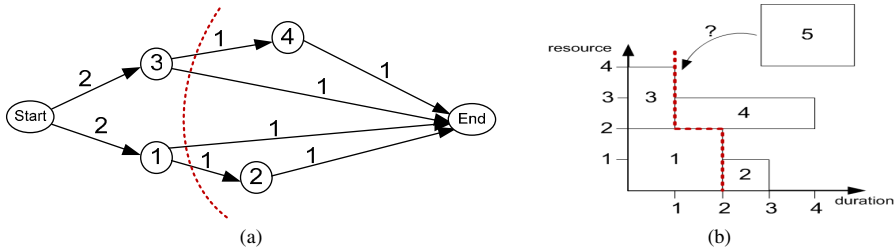


FIGURE 4. (a) A timed flow (the dotted line shows the "cut") – (b) The Gantt chart associated to timed flow (a).

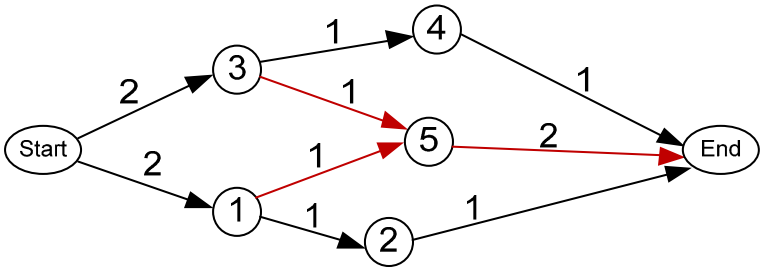


FIGURE 5. The timed flow of Figure 4 after insertion of activity 5.

Q-valued functions Out and In, with respective domains A and B, in such a way that:

$Out \geqslant 0$; $In \geqslant 0$; $\sum_{x\in A} Out(x) = \sum_{y\in B} In(y)$.

Then we say that a Q-valued vector $G = (G_{x,y} \geqslant 0,\ x \in A,\ y \in B) \geqslant 0$, defines a *match flow vector* related to those data iff:

- for any x in A, $Out(x) = \sum_{y\in B} G_{x,y}$;
- for any y in B, $In(y) = \sum_{x\in A} G_{x,y}$.

For such a *match flow* vector G, we set: $M\text{-}Makespan(G) = \mathrm{Sup}_{x\in A, y\in B \text{ such that } (x,y)\in E \text{ or } Gx,y\neq 0}(\Pi(x) + \Pi^*(y))$.

This definition leads us to introduce the following *Match Flow Problem*:
The Match Flow Problem: {*Given (X, E), A, B, In, Out, Π, Π* as above, find a related match flow vector G in such a way that M-Makespan(G) be the smallest possible.*}

Explanation: if we refer to the insertion mechanism which motivates the *Match Flow* concept, we see that if $\boldsymbol{I} = (V, K, R, r, d, \ll)$ is some RCPSP instance, if $W \subset V$ is some *Inserted Activity* subset, if F is some no circuit r^*-multi-commodity-flow vector F defined on the *Activity Network* $\boldsymbol{N}(W)$, if T and T^* are

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

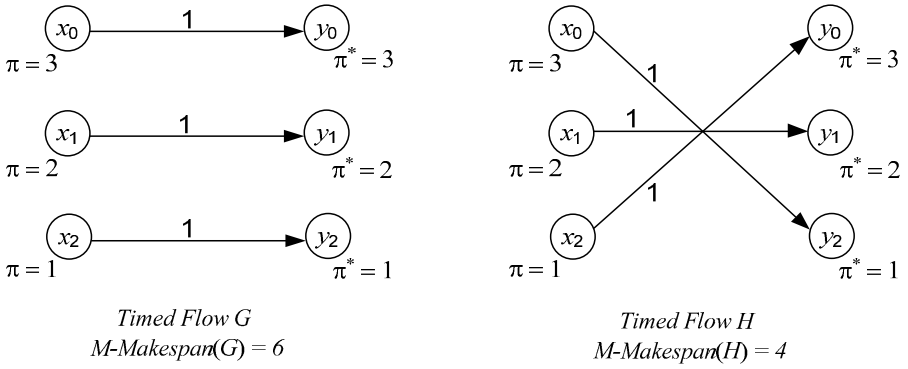


FIGURE 6. Timed flow G does not satisfy the no cross property while Timed flow H does it.

two Q-valued vectors T and T^*, defined as in (P5) and both with indexation on W^*, if $U \subset W$ defines a *Cut*, *i.e.* a partition of W into two subsets U and $W - U$, such that no flow amount goes from to $(W - U) \cup \{End\}$ to $U \cup \{Start\}$, and if $k \in K$ is some resource, then optimizing the flow values $F(k)_{(v,w)}$, $v \in U \cup \{Start\}$, $w \in (W - U) \cup \{End\}$, means solving the *Match Flow* instance defined by:

- $A = U \cup \{Start\}$; $B = (W - U) \cup \{End\}$; $X = A \cup B$; $E = (v, w)$, $v \in A$, $w \in B$, such that $vTr(\ll)w$;
- for any v in U, $\Pi(v) = T_v + d_v$; $T(Start) = 0$;
- for any v in $W - U$, $\Pi^*(v) = T^*_v$;
- for any v in $U \cup \{Start\}$, $\mathrm{Out}(v) = \sum_{w \in B} F(k)_{(v,w)}$;
- for any $w \in (W - U) \cup \{End\}$, $\mathrm{In}(w) = \sum_{v \in A} F(k)_{v,w}$.

The following *No Cross* property is going to provide us with a sufficient optimality condition for the *Match Flow* problem: G satisfies the *no cross* property if there does not exist x, $x' \in A$, y, $y' \in B$, such that:

- $G_{x,y} \neq 0$; $G_{x',y'} \neq 0$;
- $\Pi(\mathrm{x'}) > \Pi(x)$; $\Pi^*(y') > \Pi^*(y)$.

Theorem 3.1. *Any match flow vector G related to (X, E), A, B, In, Out, Π, Π^* above, and which satisfies the no cross property is an optimal solution of the Match Flow problem.*

Proof. Let us consider some *match flow* vector G, related to the above data and which satisfies the *no cross* property, and let us assume that there exists an other *match flow* vector H, related to the same data, and which is such that *M-Makespan*$(H) <$ *M-Makespan*(G). Clearly, the value *M-Makespan*(G) must be

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

TABLE 2. Example of RCPSP instance with 7 activities and 1 resource of capacity 6.

Activities	Duration	Resource requirement	Predecessors
1	6	1	–
2	3	3	–
3	4	2	–
4	2	3	–
5	2	2	–
6	3	1	–
7	5	3	–

larger than $\mathrm{Sup}_{x\in A, y\in B \text{ such that } (x,y)\in E}\ (\Pi(x) + \Pi^*(y))$. So there must exist $x \in A$, $y \in B$, such that: □

- $G_{x,y} \neq 0$;
- *M-Makespan*$(G) = \Pi(x) + \Pi^*(y)$;
- $H_{x',y'} = 0$ for any pair (x', y') such that $\Pi(x') + \Pi^*(y') \geqslant$ *M-Makespan*(G).

The difference $G - H$ induces the existence of some elementary cycle $\Gamma = (x_0, y_0, x_1, y_1, \ldots, x_n = x_0)$ such that:

- $x_0 = x$; $y_0 = y$;
- for any $i = 0 \ldots n\text{-}1$, $G_{xi,yi} > H_{xi,yi}$ and $G_{xi+1,yi} < H_{xi+1,yi}$ (see Fig. 6 for an example with $n = 3$).

Since *M-Makespan*$(H) <$ *M-Makespan*(G), we must have $\Pi(x_1) < \Pi(x_0)$, and since G is *no cross*, we must have $\Pi^*(y_1) \geqslant \Pi^*(y_0)$. Keeping on this way, we see that, for any $i = 1 \ldots n$, we must have $\Pi(x_i) < \Pi(x_0)$ and $\Pi^*(y_i) \geqslant \Pi^*(y_0)$. We get a contradiction when $i = n$. *End-Theorem.*

Clearly, the following ***Match-Flow*** Procedure provides us with such a *no cross match flow* vector:

Match-Flow Procedure

Compute 2 linear orderings $\ll_\Pi$ and $\ll_{\Pi^*}$ of A and B, respectively compatible with increasing values $\Pi(v)$ and decreasing values $\Pi^*(v)$;
$x \leftarrow \mathrm{Min}(A, \ll_\Pi)$; y $\leftarrow \mathrm{Min}(B, \ll_{\Pi^*})$;
While *x and y are both defined* **do**
  $r \leftarrow \mathrm{Inf}(\mathrm{Out}(x), \mathrm{In}(y))$; $G_{x,y} \leftarrow r$;
  $\mathrm{In}(y) \leftarrow \mathrm{In}(y) - r$; $\mathrm{Out}(x) \leftarrow \mathrm{Out}(x) - r$;
  If $\mathrm{Out}(x) \neq 0$ **then** $y \leftarrow \mathrm{Succ}(y, B, \ll_{\Pi^*})$
  Else
    $x \leftarrow \mathrm{Succ}(x, A, \ll_\Pi)$;
  Endif
EndWhile
Return *M-Makespan*(G);

Computational complexity of the *Match-Flow* procedure

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

TABLE 3. Example of RCPSP instance with 3 activities and 2 resources.

Activities	Duration	Resource requirement R1	Resource requirement R2	Predecessors
1	3	3	2	–
2	5	1	0	–
3	2	2	5	–

TABLE 4. RCPSP-Greedy-Flow procedure, *Monte-Carlo* Scheme, Mean Results.

N-Activity	*N-res*	*N-rep*	*Time* (s)	*Gap*(%) TB	*Gap*(%) LB
30	4	100	0.63	1.87	1.87
30	4	1000	6.3	0.92	0.92
30	4	5 000	31.6	0.53	0.53
30	4	50 000	317.4	0.28	0.28
60	4	100	4.54	16.91	7.10
60	4	1000	53.04	15.37	5.79
60	4	5000	243	14.56	5.13
60	4	50000	2432	13.77	4.47
120	4	100	29.6	52.33	21.32
120	4	1000	515	48.84	18.5
120	4	5000	2608	47.05	17.12
120	4	50 000	25 961	45.07	15.54

TABLE 5. RCPSP-L5-Flow procedure, Mean Results.

N−Aivity	*N-Rep*	*Strategy*	*Gap* (%) TB	*Gap* (%) LB	*Time*(s)	*Up* (%)
30	10	Crit-Path	0.85	0.85	3.77	10.07
30	10	Antichain	0.36	0.36	1.64	10.45
30	50	Crit-Path	0.49	0.49	19.41	9.85
30	50	Antichain	0.12	0.12	8.42	10.68
60	10	Crit-Path	14.34	4.97	20.11	13.11
60	10	Antichain	12.70	3.82	9.33	11.17
60	50	Crit-Path	13.69	4.4	101.50	13.04
60	50	Antichain	11.93	3.34	46.93	I1.25
120	10	Crit-Path	45.41	15.80	210.80	13.11
120	10	Antichain	38.67	12.41	63.73	16.72
120	50	Crit-Path	43.71	14.45	1055.90	13.04
120	50	Antichain	36.60	11.34	321.13	17.04

The while loop has a complexity $O(|A|+|B|)$. The computation of the 2 linear orderings at the beginning of the procedure has a complexity $O(|A|\log|A|+|B|\log|B|)$ since sorting p numbers can be performed in $O(p.\log(p))$. Nevertheless let us notice that in practice the computation of the linear orderings of A and B is performed in an incremental way, which means that the computational costs related to the sorting of A and B has not to be taken into account. Moreover, if we think into the way the ***Match-Flow*** algorithm is going to be used in order to deal

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

with a RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$, we see that cardinalities of A and B are usually going to be much smaller than the total number of activities $|V|$. As a matter of fact, those cardinalities will most often remain bounded, or increase in a logarithmic way as a function of $|V|$. This means that, practically, it will be possible to consider the running time induced by a ***Match-Flow*** call inside our global RCPSP resolution process as bounded by some constant number or by some $\mathrm{Log}(|V|)$ term.

If we link this *Match Flow* problem with the insertion of some activity v_0 into some timed (r^*, d^*)-flow through some *Cut* $W = A \cup B$, as previously explained, it comes that we should be interested, when dealing with the *Match Flow* problem, in making the resulting *match flow* G be, the most often possible, null on the arcs which are not in E. In order to put this in a formal way, we denote by E^C the set $A.B - E$, and we order the set E^C by first computing 2 linear orderings $\ll_\Pi$ and $\ll_{\Pi*}$ of A and B, respectively compatible with increasing values $\Pi(v)$ and decreasing values $\Pi^*(v)$, and by next setting:

$$(x,y)\,\sigma\left(x',y'\right) \text{iff} \begin{cases} \Pi(x)+\Pi^*(y) < \Pi\left(x'\right)+\Pi^*\left(y'\right), \\ \textbf{or } \left(\Pi(x)+\Pi^*(y)=\Pi\left(x'\right)+\Pi^*\left(y'\right) \text{ and } x \ll_\Pi x'\right), \\ \textbf{or } \left(\Pi(x)+\Pi^*(y)=\Pi\left(x'\right)+\Pi^*\left(y'\right) \text{ and } x=x' \text{and } y \ll_{\Pi^*} y'\right). \end{cases}$$

It is possible to associate with this linear ordering σ of E^C a lexicographic ordering $Lex(\sigma)$ which is defined on the E^C-indexed vectors by:

- $G\ Lex(\sigma) H$ iff there exists $(x_0, y_0) \in E^C$ such that:
  - $G_{x,y} = H_{x,y}$ for any pair (x, y) in E^C such that $(x, y)\,\sigma\,(x_0, y_0)$;
  - $G_{x0,y0} < H_{x0,y0}$.

The meaning of this lexicographic ordering σ is that the highest is some arc (x,y) according to the σ hierarchy, the less we want it to support non null match flow value $G_{x,y}$. This leads us to deal with the following problem, which expresses the fact that we are going to compute the *Match Flow* G in such a way that it minimizes the *M-Makespan* value while involving, as much as possible, support arcs which either are in E or are low ranked according to the σ linear ordering:

The Lexicographic Match Flow Problem: {*Given (X, E), A, B, In, Out,* Π *and* $\Pi^*, \ll_\Pi, \ll_{\Pi^*}$, *as above, find a related match flow vector G in such a way that:*

- *M-Makespan(G) be the smallest possible*;
- *The restriction of G to* E^C *is minimal for the above defined lexicographic order* $Lex(\sigma)$, *among all G which make M-Makespan(G) be the smallest possible*}.

In order to deal with this problem, we only need to apply the following algorithm, which iteratively minimizes (through the ***Redirection*** procedure below) the flow value G on the arcs of E^C, while dealing first with those which are the highest according to the σ hierarchy:

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

Lexicographic-Match-Flow Procedure

Apply the ***Match-Flow*** Procedure, while considering that $\ll_{\Pi}$ and $\ll_{\Pi^*}$, of this procedure are provided as part of the input of the *Lexicographic Match Flow* Problem: let G be the resulting *match flow*;
$E_{\sup} = (x_{\sup}, y_{\sup}) \leftarrow$ Largest (for the σ ordering) element (x, y) in E^C which is such that $G_{x,y} \neq 0$;
$L \leftarrow \{(x, y) \in E^C$ such that $(x, y)\sigma^{=} (x_{\sup}, y_{\sup})\}$;
While L *is not empty* **do**
    $E_0 = (x_0, y_0) \leftarrow$ Largest element of L for the σ ordering;
    Remove e_0 from L; ***Redirection*** (e_0);
EndWhile
Return M-*Makespan*(G);

Redirection Procedure

Input: (x_0, y_0)
Stop $\leftarrow$ False;
While *Stop* = *False* **do**
    Search for a sequence (simple path search) $\Gamma = (x_0, y_0, x_1, y_1, \ldots, x_n = x_0)$ such that for any $i = 1 \ldots n-1$:
        • $G_{xi,yi} \neq 0$;
        • $(x_{i+1,yi})$ is in $E \cup L$;
    If Γ *does not exist* **then** Stop $\leftarrow$ True;
    Else
        $\Delta \leftarrow \mathrm{Inf}_{i=0..n-1} G_{xi,yi}$;
        For any $i = 0 \ldots n$-1 **do**
            $G_{xi,yi} \leftarrow G_{xi,yi^-} \Delta$; $G_{xi+1,yi} \leftarrow G_{xi+1,yi} + \Delta$;
        EndFor
        If $G_{x0,y0} = 0$ **then** *Stop* $\leftarrow$ True **Endif**;
    Endif
EndWhile

Theorem 3.2. *The above* ***Lexicographic-Match-Flow*** *procedure computes an optimal solution of the Lexicographic Match Flow problem.*

Proof-Theorem. Left to the reader. *End-Theorem.*

Computational complexity of the Lexicographic-Match-Flow procedure

The theoretical complexity of the **Lexicographic-Match-Flow** procedure is quite high: the main loop can have up to $|A|^*|B|$ iterations; the **Search** instruction of the redirection process is in $O(|A|+|B|)$ (it is a simple path search in a graph); the while loop of the redirection procedure has the same complexity as an algorithm for a maximum flow problem: $O(|A|^*|B|)$. The global complexity is then $O(((|A|^*|B|)^2)\ ^*(|A|+|B|))$. Still, one checks that in practice the 2 while loops terminate very quickly. Also, if we think into our RCPSP application context, we should take into account that the cardinalities of A and B tend to be almost constant as $|V|$ increases.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

Match Multi-Commodity Flows

We consider now the same problem as above, while trying to deal with a whole resource set H. That means that Out and In are **Q-**valued functions with respective domains $H.A$ and $H.B$, where H is a resource set. It comes that, for any $h \in H$, Out(h) and In(h) respectively denote **Q-**valued functions with domains A and B. Thus, the *match flow* vector G becomes a *match multi-commodity flow* vector $G = (G(h),\ h \in H)$, and related *Match Multi-Commodity Flow* Problems may be defined in a natural way. The simple *Match Flow* problem may be handled through independent applications of the ***Match-Flow*** procedure to the various components of G. If we want to make G be the most often possible null outside the arc set E, we must proceed in such a way that, when we deal with the computation of some flow vector $G(h_0)$, once flow vectors $G(h)$, $h \in H$ have already been computed, the flow value $G_{x,y}$ be the most possible null outside the arcs (x, y) which support the flow vectors $G(h)$, $h \in H$. That means that we must adapt the ***Lexicographic-Match-Flow*** Procedure in the following way:

MF-Lexicographic-Match-Flow Procedure

$E_{Aux} \leftarrow E$;
For any $h \in H$ do G $\leftarrow 0$;
For $h \in H$ **do**
  Compute $G(h)_{x,y}$ values, $x \in A$, $y \in B$, through application of the ***Lexicographic-Match-Flow*** procedure to (X, E_{Aux}), A, B, Out(h), In(h), Π, Π^*, $\ll_{\Pi}$, $\ll_{\Pi^*}$;
  $E_{Aux} \leftarrow E_{Aux} \cup \{(x, y),\ x \in A,\ y \in B,$ such that $G(h)_{x,y} \neq 0\}$
EndFor

3.2. The Insertion Flow Problem

Insertion Flows

Since our ultimate goal is to provide an insertion flow mechanism in order to deal with timed (r^*, d^*)-flows, we are led in a natural way to introduce a notion *Insertion Flow*. We say that an oriented graph $N = (X, E)$ is *almost-bipartite*, if there exists some node z_0 in X such that the restriction of N to $X - \{z_0\}$ is bipartite, which means that $X - \{z_0\}$ may be written as the disjoint union $X - \{z_0\} = A \cup B$, of two disjoint independent sets A and B. Let us suppose now that we are endowed with two positive (or null) **Q-**valued functions Π, Out, both with domain A, with two positive (or null) **Q-**valued functions Π^*, In, both with domain B, with two positive (or null) coefficients (respectively called the duration and resource requirements of z_0) d and ρ, such that: $\sum_{x \in A}$ Out$(x) = \sum_{y \in B}$ In$(y) \geqslant \rho$. Then we say that a vector $G = (G_{x,y} \geqslant 0,\ x \in A \cup \{z_0\},\ B \cup \{z_0\}) \geqslant 0$, is an *Insertion Flow vector* related to those data: (X, E), z_0, A, B, In, Out, Π, Π^*, ρ, d, iff:

- for any x in A, Out$(x) = \sum_{y \in B \cup \{z0\}} G_{x,y}$;
- for any y in B, In$(y) = \sum_{x \in A \cup \{z0\}} G_{x,y}$;

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

- $\rho = \sum_{x \in A} G_{x,z0} = \sum_{y \in B} G_{z0,y}$.

For such an *Insertion Flow* vector G, we set:

- *Makespan1*$(G) = \mathrm{Sup}_{x \in A,\ y \in B \text{ such that } (x,y) \in E \text{ or } Gx,y \neq 0} (\Pi(x) + \Pi^*(y))$;
- *Makespan2*$(G) = \mathrm{Sup}_{x \in A,\ y \in B \text{ such that } ((x,z0) \in E \text{ or } Gx,z0 \neq 0) \text{ and } ((z0,y) \in E \text{ or } Gz0,y \neq 0)} (\Pi(x) + \Pi^*(y) + d)$;
- *I-Makespan*$(G) = \mathrm{Sup}($*Makespan1*$(G),$ *Makespan2*$(G))$.

This definition leads us to set the following *Insertion-Flow Problem*:

The Insertion-Flow Problem: {*Given (X, E),* z_0*, A, B,* In, Out*,* Π*,* Π^**,* ρ*,* d *as above, find a related Insertion Flow vector G in such a way that I-Makespan(G) be the smallest possible*}.

Explanation: if we refer to the insertion mechanism which motivates the *Insertion Flow* concept, we see that if $\boldsymbol{I} = (V, K, R, r, d, \ll)$ is some RCPSP instance, if $W \subset V$ is some *Inserted Activity* subset, if F is some no circuit r^*-multi-commodity flow vector F defined on the *Activity Network* $\boldsymbol{N}(W)$, if T and T^* are two **Q-**valued vectors T and T^*, defined as in (P5) and both with indexation on W^*, if $U \subset W$ defines a *Cut*, *i.e.* an ad hoc partition of W into two subsets U and $W - U$, if $k \in K$ is some resource, if v_0 is some activity in $V - W$ which is to be inserted, then optimizing the flow values $F(k)_{(\mathrm{v,w})}$, $v \in U \cup \{Start\}$, $w \in (W - U) \cup \{End\}$, $F(k)_{(\mathrm{v,v0})}$, $v \in U \cup \{Start\}$, $F(k)_{(\mathrm{v0,w})}$, $w \in (W - U) \cup \{End\}$, means solving an *Insertion Flow* instance defined by:

- $A = U \cup \{Start\}$; $B = (W - U) \cup \{End\}$; $X = A \cup B$;
- $E = (v, w)$, $v \in A$, $w \in B$, such that $v\ \mathrm{Tr}(\ll)w$;
- $z_0 = v_0$; $\rho = r_{k,v0}$; $d = d_{v0}$;
- for any v in U, $\Pi(v) = T_v + d_v$; $T(Start) = 0$; for any v in $W - U$, $\Pi^*(v) = T^*_v$;
- for any v in $U \cup \{Start\}$, $\mathrm{Out}(v) = \sum_{w \in B} F(k)_{(v,w)}$;
- for any $w \in (W - U) \cup \{End\}$, $\mathrm{In}(w) = \sum_{v \in A} F(k)_{(v,w)}$.

Remark: Feasibility of the *Insertion-flow* Problem

According to this interpretation, we understand that the *Insertion-Flow* problem has always a solution since:

- the number $\sum_{x \in A} \mathrm{Out}(x)$ of resources given by A is equal to the number $\sum_{y \in B} \mathrm{In}(y)$ of resources required by B and is at least equal to the number ρ of resources which are required by task z_0;
- the arcs in E are related to precedence relations between the activities of A and the activities of $B \cup \{z_0\}$ or between z_0 and the activities of B (there is precedence relation neither from B to A nor from B to z_0).

It comes that a disjunction relationship between an activity x in A (B) and z_0 (which derives from the fact that the sum of the resources required by those 2 activities is more than $R = \sum_{x \in A} \mathrm{Out}(x)$ means that the activity x will have to

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

give (receive) some resources to (from) z_0 and so, that x will precede (succeed) z_0 in the resulting schedule.

In order to deal with this *Insertion Flow* problem, we first build (pre-treatment) 2 linear orderings $\ll_\Pi$, $\ll_{\Pi^*}$, of A and B, respectively compatible with increasing values of Π and with decreasing values of Π^*. Next, we notice that our *Insertion Flow* problem is resolved once we have identified the *Attachment Node* u, *i.e.* the node $x \in A$, such that:

- $G_{u,z0} \neq 0$;
- For any $x \in A$ such that: $u \ll_\Pi x$ then we have: $G_{x,z0} = 0$.

As a matter of fact, if the *attachment node* u is known, we may compute the values $G_{x,z0}$, $x \in A$, through an ***Attach*** Procedure, in such a way that: (P6)

- if x and x' are such that: $x' \ll_\Pi x$ $(\ll_{\Pi=})u$ and $G_{x',z0} \neq 0$, then we have: $G_{x,z0} = \mathrm{Out}(x)$;
- for any x such that: $u \ll_\Pi x$ then we have: $G_{x,z0} = 0$.

The procedure ***Attach*** is entirely determined by (P6), and consequently modifies the values $\mathrm{Out}(x)$, $x \in A$. Once ***Attach***(u) has been performed, we only need to add z_0 to the set A and to apply the ***Match-Flow*** procedure in a convenient way, in order to get an *Insertion Flow*. This may be summarized through the following ***Try-Insertion*** algorithm:

Try-Insertion Procedure

Input: u (a node of A)
Attach(u);
Extend Π, Out and $\ll_\Pi$ to $A \cup \{z_0\}$, in such a way that:
- $\Pi(z_0) = \mathrm{Sup}_{x \in A((x,z0) \in E \text{ or } Gx,z0 \neq 0)}(\Pi(x) + d)$;
- $\ll_\Pi$ becomes defined on $A \cup \{z_0\}$, and compatible with increasing values $\Pi(x)$;
- $\mathrm{Out}(z_0) = \rho$

Apply the ***Match-Flow*** procedure to (X, E), $A \cup \{z_0\}$, B, In, Out, Π, Π^* while considering that $\ll_\Pi$ and $\ll_{\Pi^*}$ of this procedure are provided as results of the above instructions;
Return *I-Makespan*(G);

As for the search for a convenient attachment node u, we handle it in an exhaustive way by scanning what we call the *relevant subset* $A_{E,z0}$ of A, and which is defined by:

$A_{E,z0} = \{x \in A$, such that:

- there does not exist $x' \in A$, such that $x \ll_\Pi x'$ and $(x', z_0) \in E$;
- $\sum_{x' \in A,\, x' (\ll_\Pi \mathrm{or} =)\, x} \mathrm{Out}(x') \geqslant \rho\}$.

That means that our *Insertion-Flow* problem may be handled through the following ***Insertion*** Procedure:

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

Insertion Procedure
Compute 2 linear orderings $\ll_\Pi$ and $\ll_{\Pi^*}$ of A and B, respectively compatible with increasing values $\Pi(v)$ and decreasing values $\Pi^*(v)$; **Compute** $u_0 \in A_{E,z0}$ defined as above, in such a way that the *Try-Insertion* value S_0 provided by an application of the procedure ***Try-Insertion*** (u_0) be the smallest possible; ***Return*** (u_0, S_0);

Comment: *Insertion* provides us with a pair (attach-node u_0, *I-Makespan* value S_0).

Computational complexity of the *Insertion* procedure

Running the ***Insertion*** procedure mainly means scanning the possible attachment nodes u and, for every such a node u, performing the ***Try-Insertion*** procedure that is first performing the ***Attach*** procedure and next applying the ***Match-Flow*** procedure to the resulting *Match-Flow* instance. The complexity of this last combination of processes is $O(|A|+|B|)$. It comes that the complexity of the whole ***Insertion*** procedure is $O(|A|.\ (|A|+|B|))$.

We may state:

Theorem 3.3. *The* ***Insertion*** *procedure yields an optimal solution of the Insertion Flow Problem.*

Proof. In order to prove this statement, we first need to check that if we consider some feasible solution G of the *Insertion Flow* Problem, and if we set:

$u =$ Largest, according to the ordering $\ll_\Pi$, element $x \in A$, such that $(x, z_0) \in E$ or $G_{x,z0} \neq 0$, □

then it is possible to modify G in such a way that we end getting the (P6) property related to u, without making us lose the feasibility of G and without deteriorating *I-Makespan*(G). Clearly, once the (P6) property will be satisfied by u and G, u will be the attachment node, and also that u will belong to the relevant subset $A_{E,z0}$. In order to do it, we suppose that u and G do not satisfy (P6) and we consider for instance x, x' in A, such that: $x \ll_\Pi x'$ $(\ll_{\overline{\Pi}}^{=})u$; $G_{x',z0} \neq 0$ and $G_{x,\,z0} \neq \mathrm{Out}(x)$. Then we choose $y \in B$, such that $G_{x,y} \neq 0$, and we apply to G the following flow redirection process:

- $\delta \quad \leftarrow \mathrm{Inf}\ G_{x',z0}, G_{x,y}$;
- $G_{x',z0} \leftarrow G_{x',z0} - \delta$; $G_{x,y} \leftarrow G_{x,y} - \delta$; $G_{x',y} \leftarrow G_{x',y} + \delta$; $G_{x,z0} \leftarrow G_{x,z0} + \delta$.

Clearly, applying this redirection process makes G get closer to the (P6) property while maintaining it as a feasible solution of the *Insertion Flow* Problem. Also, we notice that the value *I-Makespan*(G) has not increased. Thus, if G is an optimal solution of the *Insertion Flow* Problem, we may turn it into an optimal solution G' which satisfies (P6).

Concluding the proof is easy, since Theorem 3.1 tells us that we may next deduce, through application of the ***Match-Flow*** Procedure, an *Insertion Flow* G''

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

which fits the statement of Theorem 3.3 and which is such that *I-Makespan*$(G'') \leqslant$ *I-Makespan*(G'). *End-Theorem.*

Insertion Multi-Commodity Flows

The above *Insertion* algorithmic scheme can be easily adapted to the case of multi-commodity flow vectors. We only need to notice that the same attachment node may be used for all the *Insertion Flow* vectors $G(h)$, $h \in H$, H being the commodity set, and that this node needs to be in the above defined subset $A_{E,z0}$ of A. Also, if part of the goal is to make in such a way that the number of arcs in E^C which carry non null multi-commodity flow values be the smallest possible, it is possible to derive a **MF-Lex-Insertion Procedure** (u: u is a node of the subset $A_{E,z0}$ of A) from the ***Lexicographic-Match-Flow*** procedure which we just previously described.

3.3. Difference between our insertion mechanism and Artigues *et al.* one

As told at the beginning of part 3, our algorithms may be compared with those which have been proposed in [6, 7]: the basic difference lays upon the fact that every time the insertion of some activity is performed, it involves the resolution of a specific *Insertion Flow* sub-problem related to a *Cut* of the currently inserted activity set; so, the related resolution process updates all the flow values which express the flow transportation between both sides of this *Cut*, while in Artigues *et al.* proposal, the *insertion* process does not involve any *Cut*, but a specific arc subset E: part of the flow which runs along an arc $[x, y]$ of E is redirected along the arcs $[x, z_0]$ and $[z_0, y]$ in order to first provide with resource coming from x, before giving back this resource to y.

This main difference between the two methods is highlighted in the following Figures 7 and 8 related to the instance described in Table 2: activity 7 is to be inserted into the cut (1,2,3)/(4,5,6). The initial flow (before insertion of activity 7) is given by the Figure 7a.

Figure 7b shows the flow after the insertion of the activity 7 using the mechanism proposed in [7] while Figure 7c shows the flow after the insertion of the activity 7 using our method. In the two figures, the new (or changed) flows are in dot line and the Gantt charts associated to each solution are shown in Figure 8. We see that our method, which allows to change all the flow in the cut, leads to a new makespan of 10 while [7] method leads to a new makespan of 12.

4. Generic Flow Algorithms for the RCPSP

4.1. A Greedy Insertion Algorithm

This algorithm ***RCPSP-Greedy-Flow*** works as follows: at any time while processing some RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$, it deals with some *Inserted Activity* subset W of V, together with a no circuit r^*-multi-commodity

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

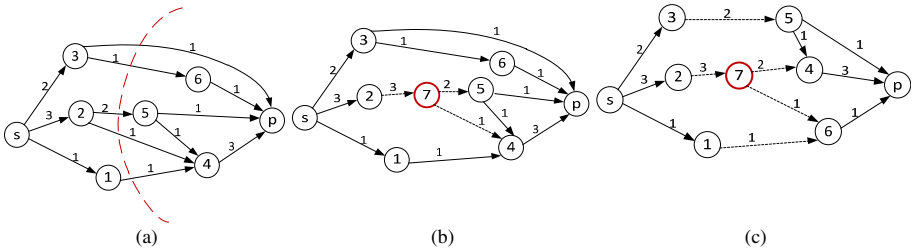


FIGURE 7. (a) Partial solution – (b) Insertion of activity 7 in the cut (1,2,3)/(4,5,6) using the mechanism proposed in [7] – (c) Insertion of activity 7 in the cut (1,2,3)/(4,5,6) using our mechanism.

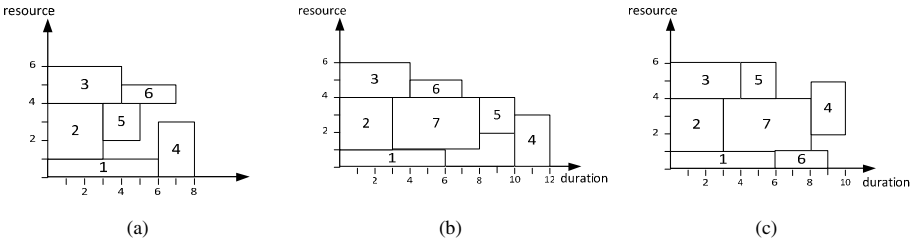


FIGURE 8. (a) Gantt chart related to the initial solution – (b) Gantt chart related to the flow of Figure 7b (using the insertion mechanism proposed in [7]) – (c) Gantt chart related to the flow of Figure 7c (using our proposed mechanism).

flow F defined on the *Activity Network* $\boldsymbol{N}(W)$, with two positive (or null) Q-valued vectors T and T^*, both with indexation on $V^* = V \cup \{Start, End\}$, and with 2 linear orderings $\ll_T$ and $\ll_{T^*}$, respectively compatible with increasing values $T_v + d_v$ (with $d_{Start} = 0$) and decreasing values T^*_v, in such a way that, for any v in W:

- T_v = Length of a largest path, in the sense of the d* length vector, from *Start* to v in the *Support Partial Activity Network* defined by $E(F, \ll)$;
- T^*_v = Length of a largest path from v to *End* in the *Support Partial Activity Network* defined by $E(F, \ll)$.

Then it randomly picks up some activity v_0 in $V - W$, and it "inserts" it into the timed (r^*, d^*)-flow (F, T), *i.e.* it turns F into a convenient no circuit r^*-multicommodity flow defined on the *Activity Network* $\boldsymbol{N}(W \cup \{v_0\})$. That means that it selects a *Cut* of (W, F), that means some subset U of W such that: (P7)

- for any v in U, and any v' in W such that ($v' \ll v$ or $F_{(v',v)} \neq 0$), we have $v' \in U$;

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

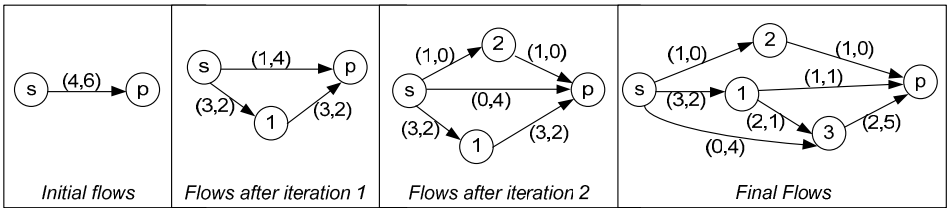


FIGURE 9. Successive insertion of activities related to the instance of Table 3.

- for any $v \in W$, such that $v\ Tr\ (\ll)v_0$, we have $v \in U$, and for any $v \in W$, such that $v_0\ Tr\ (\ll)v$, we have $v \in W - U$.

For such a well-chosen Cut U, it sets: (P8)

- $X = W \cup \{Start, End, v_0\}$; $A = U \cup \{Start\}$; $B = X - A - \{v_0\}$; $z_0 = v_0$;
- $d = d_{v_0}$; $H = K$; for any k in $K = H$, $\rho(k) = r_{k,v0}$;
- $E = \{(x, y), x \in A \cup \{v_0\}, y \in B \cup \{v_0\}$, such that $x\ Tr\ (\ll)y\ \}$;
- for any x in $U \cup \{Start\}$, any k in K, $\mathrm{Out}(k, x) = \sum_{y \in B} F(k)_{(x,y)}$;
- for any y in $X - (U \cup \{Start, z_0\})$, any k in K, $\mathrm{In}(k, y) = \sum_{x \in A} F(k)_{(x,y)}$;
- for any x in $A = U \cup \{Start\}$, $\Pi(x) = T_x + d_x$ (with $d_{Start} = 0$) ; for any x in B, $\Pi^*(x) = T^*_x$;
- $\ll_{\Pi} = \ll_{T}$; $\ll_{\Pi^*} = \ll_{T^*}$.

This construction makes possible the call to the ***MF-Insertion*** and the ***MF-Lex-Insertion*** Procedures we just described in 3.2, which yield an *Insertion Multi-commodity-flow* vector G. Then ***RCPSP-Greedy-Flow*** updates the timed (r^*, d^*)-flow (F, Π) by setting: (P9)

- for any k in K, v in $U \cup \{Start, v_0\}$, w in $(W - U) \cup \{End, v_0\}$: $F(k)_{(v,w)} = G(k)_{(v,w)}$,

by keeping on with former values $F(k)_{(v,w)}$ for any pair (v, w) in $A.A$ or in $B.B$, and by updating every value T_v, $v \in W \cup \{v_0\}$, as the length, for the length vector d^*, of a largest path from *Start* to v in the *Support Partial Activity Network* defined by $E(F, \ll)$. Clearly, the definition of a Cut (property (P7)) keeps the arc set $E(F, \ll)$ from defining any circuit, and (F, T) remains a timed (r^*, d^*)-flow.

The Figure 9 shows a full example: we consider an instance with 3 activities and 2 resources and without precedence constraint (see Tab. 3). First resource has a capacity 4 and second resource has a capacity 6.

Searching for a best Cut U in the general sense seems to be a difficult problem. As a matter of fact, we may state:

Theorem 4.1. *The search for a best Cut as defined above is NP-Complete.*

Proof

Let us consider the following situation, which provides us with an input for the *Insertion Cut* Problem:

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

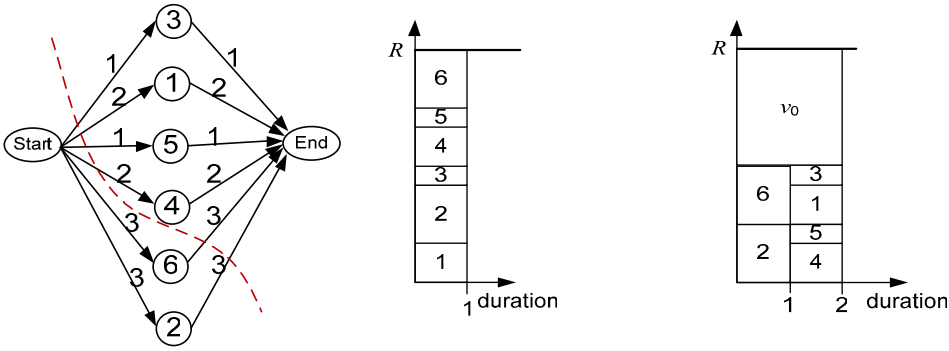


FIGURE 10. Left: timed flow for 6 activities with duration 1, the optimal cut for insertion of v_0 is shown in dotted line – Center: Gantt chart which represents the timed flow left – Right: Gantt chart after insertion of v_0 in the cut.

- there is only one resource (so we set $r_{k,x} = r_x$) and $R_k = R$, for the unique k in K;
- $\sum_{v \in W} r_v = R$; $r_{v0} = R/2$; $\ll$ is the empty relation; for any v in W, $d_v = 1$; $d_{v0} = 2$.

The current flow F must be the trivial flow defined by: for any v in W, $F_{(\text{Start,v})} = r_v = F_{(\text{v,End})}$, which induces a *Makespan* value equal to 1. Clearly, determining whether the optimal value of the *Insertion Cut* instance is equal ($\leqslant$) to 2 means solving an instance of the *2-Partition* problem (see Fig. 10). *End-Theorem.*

Still, if we consider now some node v of $W \cup \{End\}$, such that:

- for any $w \in W$, such that $w\ Tr\ (\ll) v_0$ we have $w \ll_{\mathrm{T}} v$;
- for any $w \in W$, such that $v_0\ Tr\ (\ll) w$ we have $v \ll_{\mathrm{T}} w$ or $w = v$,

then we may associate with v, in a natural way, a Cut $Cut(v)$, by setting: $Cut(v) = \{v' \in W \text{ such that } v' \ll_{\mathrm{T}} v\}$. While searching for a best $Cut\ U$ in the general sense is a difficult problem, we can easily scan the list $\ll_{\mathrm{T}}$ and choose v_1 in such a way that an application of the ***MF-Insertion*** Procedure to $U_1 = Cut(v_1)$ in the sense of (P8) and (P9) yields the best possible *Makespan* value. So, our first ***RCPSP-Greedy-Flow*** algorithm works this way, by applying the ***MF-Insertion*** procedure to a well-chosen Cut $Cut(v_1)$ as it has just been told, by next updating T, T^*, $\ll_{\mathrm{T}}$ and $\ll_{T^*}$, and by keeping on with the insertion process until all activities have been inserted. The whole process may be summarized as follows:

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

RCPSP-Greedy-Flow Algorithm

Input: the instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$ of the RCPSP problem;
Output: a timed (r^*, d^*)-flow (F, T), and the related *Makespan* value Δ
Initialization:
$W \leftarrow$ Nil; $\Delta \leftarrow 0$; $F \leftarrow 0$; $\ll_{\mathrm{T}} = \ll_{T^*} \leftarrow \{Start, End\}$;
$T(Start) = T(End) = T^*(Start) = T^*(End) \leftarrow 0$;
Main Loop:
RCPSP-Flow $\leftarrow$ *Packet-Insertion*(V);

Packet-Insertion Procedure

Input: the instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$ of the RCPSP problem, given together with a timed r^*-flow vector F, two vectors T and T^*, and two linear orderings $\ll_{\mathrm{T}}$ and $\ll_{T^*}$, related to the restriction of $(V, K, R, \ll, r, d)$ to $W = V - S$ where S is a subset of V
Output: a timed (r^*, d^*)-flow (F, T), and the related *Makespan* value Δ
$S_{\mathrm{Aux}} \leftarrow$S; $W \leftarrow V - S$;
While $S_{\mathrm{Aux}} \neq Nil$ **do**
Randomly Pick up v_0 in S_{Aux} and Remove it from S_{Aux}; **(I1)**
Compute v_1 in $W \cup \{End\}$, such that the application of the ***MF-Insertion*** procedure to: **(I2)**
- $X = W \cup \{Start, End, v_0\}$; $A = Cut(v_1) \cup \{Start\}$; $B = X - A - \{v_0\}$; $z_0 = v_0$; $d = d_{v0}$; $H = K$; for any k in K, $\rho(k) = r_{k,z0}$;
- $E = \{(x, y), x \in A \cup \{v_0\}, y \in B \cup \{v_0\}$, such that $x\ Tr\ (\ll)\ y\ \}$;
- *Out* defined by: for any x in $Cut(v_1) \cup \{Start\}$, any k in K, $\mathrm{Out}(k, x) = \sum_{y \in B} F(k)_{x,y}$;
- In defined by: for any y in $X - (Cut(v_1) \cup \{Start, z_0\})$, any k in K, $\mathrm{In}(k, y) = \sum_{x \in AF}(k)_{x,y}$;
- for any x in A$= Cut(v_1) \cup \{Start\}$, $\Pi(x) = T_x + d_x$(with $d(Start) = 0$);
- for any x in B, $\Pi^*(\mathrm{x}) = T^*_{\mathrm{x}}$;
- $\ll_{\Pi} = \ll_{\mathrm{T}}$; $\ll_{\Pi^*} = \ll_{\mathrm{T}^*}$;

yields the best possible *Makespan* value;
Let u_1 be the attachment node provided by the related application of ***MF-Insertion***;
Apply the ***MF-Lex-Insertion*** (u_1) procedure to the input related to v_1 according to the instruction **(I2)**, in order to perform the insertion of v_0 into the timed (r^*, d^*)-flow (F, T) in an effective way;
Let G be the resulting *Insertion Flow* vector: **update** F values by setting ((P9) equations), for any k in K, v in $Cut(v_1) \cup \{Start, v_0\}$, w in $(W - Cut(v_1)) \cup \{End, v_0\}$: $F(k)_{v,w} = G(k)_{v,w}$; For any $v \in W \cup \{Start, End, v_0\}$, **update** T_v (T^*_v) as the largest length (for the d^* length vector) of a path from *Start* to v (from v to *End*) in the *Support Partial Activity Network* defined by $(F, \ll)$);
Update $\ll_{\mathrm{T}} = \ll_{\mathrm{T}*}$; $\Delta \leftarrow T(End)$; $W \leftarrow W \cup \{v_0\}$;
EndWhile
Return (F, T, Δ);

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

Remark 4.2. This algorithm may be randomized and next integrated it into a *GRASP* algorithmic scheme:

GRASP RCPSP-Greedy-Flow Procedure

Input: N_{Rep} (number of replications)
For $i = 1 \ldots N_{Rep}$ **do**
    Apply the ***RCPSP-Flow*** Procedure;
EndFor
Keep the best result (F, Π, Δ) obtained;

4.2. A Local Search Algorithm

The above ***Packet-Insertion*** operator gives rise in a generic way to a local search operator. The idea is that, once we are endowed with a timed (r^*, d^*)-flow (F, T) defined on the activity set V, we may pick up some (small) subset S of V, take it away from V (which means *reversing* the insertion process) and, next, come back to inserting the activities of S into the pair (F, T). In order to describe this operator in a more accurate way, we need to introduce the *Reverse-Insertion* process. This process operates on an activity subset $W \subseteq V$, and on a timed (r^*, d^*)-flow (F, T). It takes some activity v in W as a parameter and it proceeds as follows:

Reverse-Insertion Procedure

Input: v_0
Compute T^*, as well as 2 linear ordering $\ll_{\mathrm{T}}$ and $\ll_{\mathrm{T}*}$, respectively compatible with T and T^* values;
$A \leftarrow \{v \in W \cup \{Start\}$such that $v \ll_{\mathrm{T}} v_0\}$;
$B \leftarrow \{v \in W \cup \{End\}$such that $v_0 \ll_{\mathrm{T}} v\}$;
$E \leftarrow \{(x, y), x \in A, y \in B$, such that $x\ Tr\ (\ll)\ y\}$;
Apply the ***MF-Lexicographic-Match-Flow*** procedure while considering that:

- for any $k \in K$, $v \in A$, $\mathrm{Out}(k, v) = \sum_{w \in B \cup \{\mathrm{vo}\}} F(k_{v,w})$;
- for any $k \in K$, $v \in B$, $\mathrm{In}(k, v) = \sum_{w \in A \cup \{\mathrm{vo}\}} F(k)_{w,v}$;

Let G be the resulting *Multi-commodity Match Flow* vector: for any $v \in A$, $w \in B$, $k \in K$, set $F(k)_{v,w} = G(k)_{v,w}$;
Update T;

This elementary process may be extended into a more general ***Packet-Reverse-Insertion*** procedure, which deals with an activity subset $S \subseteq V$, and which removes it from a timed (r^*, d^*)-flow (F, T):

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

Packet-Reverse-Insertion Procedure

Input:S (subset of V)
$S_{Aux} \leftarrow S$; $W \leftarrow V$;
While $S_{Aux} \neq$ Nil **do**
  Pick up v_0 in S_{Aux} and remove it from S_{Aux};
  Reverse-Insertion(v_0);
  Remove v_0 from W;
EndWhile

This enables us to define a local search operator ***Transform-Insertion***, which is going to work on any timed (r^*, d^*)-flow (F, T), related to $\boldsymbol{I} = (V, K, R, r, d, \ll)$, with parameter an activity subset $S \subseteq V$:

Transform-Insertion Procedure

Input: S (subset of V)
Packet-Reverse-Insertion(S);
Packet-Insertion(S);

Provided with this operator, we may design algorithms according to the following scheme:

RCPSP-LS-Flow Procedure

Input: N_{Try} (number of tries)
Compute, through the ***RCPSP-Greedy-Flow*** procedure, an initial timed (r^*, d^*)-flow (F, T), together with a *Makespan* value Δ;
$j \leftarrow 1$;
While $j \leqslant N_{Try}$ **do**
  Compute some subset $S \subset V$; (I3)
  Compute the *Makespan* value Δ' resulting from the application of ***Transform-Insertion***(S) to (F, T);
  If $\Delta' < \Delta$ **then**
    Replace the current (F, T) by the timed (r^*, d^*)-flow which results from the application of ***Transform-Insertion***(S) to (F, T);
    $j \leftarrow 1$;
  Else
    $j \leftarrow$j + 1;
  Endif
EndWhile

Comment: of course, it would be possible to use a *Tabu Search* scheme or a *Simulated Annealing* scheme.

Clearly, the basic instruction is here the (I3) instruction. We tried the following approaches:

- **crit-path strategy**: S is the activity subset defined by a critical path;
- **antichain strategy**: S is the activity subset $S(t)$ defined by the activities which are simultaneously run at some instant t, according to the schedule defined by the current vector T. Several strategies have been tested to choose

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

a "good" date t: for example the date at which the resources are the less used or at which there are the less activities in parallel. The strategy which provides us with the best results was to choose the date t randomly.

Remark: is a "good" insertion order likely to produce an optimal solution?

One may ask in a natural way if there always exist, for a given RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$, some insertion order σ of V, such that performing the ***RCPSP-Greedy-Flow*** process while picking up the tasks of V according to σ yields an optimal solution. Intuitively, one feels that this should be true. Unfortunately, we were not able to prove it, and solving this open question may happen to be difficult. Actually, it would be sufficient, in order to do it, to prove that if some timed-flow (F, T) is an optimal solution of $\boldsymbol{I}$, then it is possible to choose an activity v in such a way that applying the ***Packet-Reverse-Insertion*** procedure with $S=\{v\}$ leads to an optimal solution for the restriction of $\boldsymbol{I}$ to V-$\{v\}$.

5. Numerical Tests

5.1. Results about RCPSP-Flow and RCPSP-LS-Flow procedures

We performed our experiments, on PC AMD opteron 2.1 GHz, while using gcc 4.1 compiler. We tried several instance packages, all of them obtained from the PSPLIB test bed, and, for every package, we tried both ***Monte-Carlo-RCPSP-Greedy-Flow*** and procedure with several distinct values of the parameter I. For every instance package, we kept memory of the following quantities:

- *N-Activity* = the common number of activities; *N-Res* = the common number of resources;
- *N-Rep* = the parameter value for the ***Monte-Carlo-RCPSP-Flow*** procedure, *i.e.* the number of replications of the ***RCPSP-Flow*** procedure;
- *Time* = the time in seconds needed to performed the *N-rep* replications;
- *Gap* TB = the gap between the best value obtained through *N-rep* replications of the ***RCPSP-Flow*** procedure and:
  - in the case of 30 job instances, the optimal value;
  - in the case of 60 and 120 job instances, the trivial (largest$\ll$-path) lower bound value.
- *Gap* LB = the gap between the best value obtained through *N-rep* replications of the ***RCPSP-Flow*** procedure and the best known lower bound value (optimal value if $|V|= 30$):

The following table provides us with average results for the ***RCPSP-Greedy-Flow*** *and* ***RCPSP-LS-Flow*** Procedures, related to the PSPLIB packages respectively defined by the 480 instances of 30 jobs, by the 480 instances of 60 jobs and by the 600 instances of 120 jobs.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

TABLE 6. Comparison with other methods of literature.

Reference	j30			j60			j120		
	1000	5000	50000	1000	5000	50000	1000	5000	50000
[52]	0.10	0.04	0.00	11.7I	11.17	10.74	34.74	33.36	32.06
[35]	0.27	0.11	0.01	11.73	11.10	10.71	35.22	33.10	31.57
[77]	0.27	0.06	0.02	11.56	11.10	10.73	34.07	32.54	31.24
[78]	034	0.20	0.02	12.21	11.27	10.74	35.39	33.24	31.58
[5]	0.25	0.06	0.03	11.89	11.19	10.84	36.53	33.91	31.49
our heuristic	**0.36**	**0.12**	**0.03**	**12.70**	**11.93**	**11.40**	38.67	**36.60**	**35.00**
[4]	0.33	0.12		12.57	*11.86*	–	39.36	36.57	–
[76]	0.25	0.13	0.05	11.88	11.62	11.36	35.01	34.41	33.71
[64]	0.46	0.16	0.05	12.97	12.18	11.58	40.86	37.88	35.85
[74]	0.30	0.16	0.07	12.18	11.87	11.54	36.49	35.81	35.01
[40]	0.38	0.22	0.08	12.21	I1.70	11.21	37.19	35.39	33.21
[41]	0.54	0.25	0.08	12.68	11.89	11.23	39.37	36.74	34.03
[75]	0.30	0.17	0.09	12.14	11.82	11.47	36.24	35.56	34.77
[73]	0.42	0.17		12.77	12.03	–	–	–	–
[16]	0.38	0.23		12.75	11.90	–	42.81	37.68	–
[26]	0.74	0.33	0.16	13.28	12.63	11.94	39.97	38.41	36.44
[71]	0.65	0.44		12.94	12.58	–	39.85	38.70	–

The next table provides us with average results for the ***RCPSP-LS-Flow*** Procedure, related to the same PSPLIB packages. *N-Rep* means the replication number of the GRASP RCPSP-LS-Flow scheme. The value *Up* denotes the improvement which was due to the local search process in relation to the initialization through *RCPSP-Greedy-Flow*.

Comment: those results seem to be very satisfactory, taken into account the simplicity and the generic features of those algorithms which derive from our Timed Flow framework. Also, one may notice the good behaviour of the generic local search operator *Transform-Insertion*. In order to confirm this feeling we have performed a full comparison with the best known methods of literature which is described in the following section.

5.2. COMPARISON WITH OTHER METHODS OF LITERATURE

We propose in this part to give the performances of our heuristic according to evaluation method proposed by Kolish and Hartmann [49]. They rank heuristics from literature comparing the best solution obtained through a same number of schedules evaluations. Table 6 shows the gap to the optimal solution for the j30 instances and to the trivial lower bound for the j60 and j120 instances for 1000,

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

TABLE 7. Classification of the 30 job instances of the PSPLIB library.

Instance Package	*RESOURCE-RELAX*			*RESOURCE-TYPE*			*PARALLEL-MEAN*			*PARALLEL-MAX*		
	min	max	moy	min	max	moy	min	max	moy	min	max	moy
j301	0	26.3	13.8	1	1	1	2.4	3.4	3.0	4	7	5.4
j302	0	11.8	4.4	1	1	1	3.0	3.9	3.5	5	7	6
j303	0	15.2	3.1	1	1	1	2.0	3.6	2.9	4	6	5.5
j304	0	0	0	1	1	1	2.8	3.9	3.2	5	7	5.9
j305	25	76.7	38.7	2.03	2.03	2.03	2.0	3.0	2.4	3	5	4.4
j3011	0	5	1.4	3.03	3.03	3.03	2.3	4.3	3.2	4	7	5.1
j3012	0	0	0	3.03	3.03	3.03	3	4.7	3.7	4	8	6.1
j3013	31.1	121	61.2	4	4	4	1.6	2.3	2	2	3	2.9
j3014	0	20.5	7.1	4	4	4	2.7	3.7	2.9	3	5	4
j3015	0	8.7	1.2	4	4	4	2.6	3.7	3	4	6	4.6
j3021	22.6	60.5	35.3	2.03	2.03	2.03	1.8	2.6	2.2	3	4	3.8
j3022	3.3	15.6	7.9	2.03	2.03	2.03	2.3	3.6	3.0	4	7	5.2
j3023	0	8.6	2.6	2.03	2.03	2.03	2.5	3.3	2.9	4	7	5.3
j3024	0	0	0	2.03	2.03	2.03	2.5	4.1	3.2	5	7	5.7
j3025	23.4	81.6	59.9	3.03	3.03	3.03	1.67	2.44	2.0	3	3	3
j3031	0	12.2	2.9	4	4	4	2.6	3.4	2.9	3	5	4.5
j3032	0	0	0	4	4	4	2.7	4.1	3.2	5	8	5.5
j3033	0	31	14.6	1	1	1	2.3	3.2	2.8	3	6	4.7
j3034	0	16	5.3	1	1	1	2.4	3.5	2.8	4	6	4.6
j3035	0	7.3	2.0	1	1	1	2.5	3.2	2.9	4	6	5.3
j3041	33.3	81.6	60.5	3.03	3.03	3.03	1.4	2.0	1.8	2	4	3.2
j3042	0	22.4	7.4	3.03	3.03	3.03	2.0	3.1	2.6	3	5	3.9
j3043	0	3.8	1.8	3.03	3.03	3.03	2.3	3.1	2.7	4	5	4.2
j3044	0	0	0	3.03	3.03	3.03	2.5	3.7	3.2	4	7	5.3
j3045	40.3	98.4	59.4	4	4	4	1.3	1.9	1.6	2	3	2.3

5000 and 50000 schedules evaluations. This table is sorted by ascending gap value relative to j30 instances and 50000 schedule evaluations. For the authors who provided results for different methods in the same article we kept the best results.

Comment: We notice that our algorithm provide us with some of the best available results, not too far behind GA, TS-path relinking of [52], Scatter Search FBI of [35], GA FBI of [78] and GA-forward/Backward of [5].

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

5.3. Results about structural properties of the instances

We also performed a more detailed experiment, while trying to make appear the way structural properties of the instances may eventually impact the behaviour of the algorithm. In order to do it, we started by classifying those instances according to several indicators:

- *RESOURCE-RELAX*: gap between reference makespan and trivial makespan (the makespan which is induced by the relaxation of the resource constraint);
- *RESOURCE-TYPE*: average quantity of resource type required by jobs;
- *PARALLEL-MEAN*: average number of parallel jobs per instance related to an optimal solution;
- *PARALLEL-MAX*: maximum number of parallel jobs per instance related to an optimal solution;

In the case of the 25 packages j301..j305..j341..j345 which allow us to decompose the 30 job instances of the PSPLIB library into 10 instance packages, it yielded the following Table 7:

Then we applied to those instances the ***RCPSP-Greedy-Flow*** Procedure, *Monte-Carlo* Scheme, while considering with *N-Rep* = 1000, and we got the following Table 8:

Comment: it is interesting to notice the correlation which exists between the true difficulty of the instances and the parallelism level which they allow. One should be cautious while trying to interpret it. Still, it seems to be that, in most cases, a low parallelism level tends to make the instance more difficult. Indeed a low parallelism level, which creates implicit disjunction constraints, discriminates insertions in a more significant way, and makes the quality of the final schedule more sensitive to "bad" insertion decisions. Conversely, a high parallelism level makes easier to partially offset non optimal insertion decisions.

6. Conclusion

What we just did here was trying to take advantage of the existing link between Flow Theory and Resource Constrained Scheduling. Clearly, one of the focus here was genericity: we got structural results which helped us in designing greedy and local search flow based algorithms. We tested those algorithms which proved themselves to be rather efficient. It would be interesting to try to go further, and study the way other scheduling problems (involving pre-emption...) might be cast into the Flow formalism. Also, it would be interesting to find the way to take more advantage from the general Flow Theory algorithmic machinery while efficiently dealing with the combinatorial no circuit constraint.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

TABLE 8. RCPSP-Greedy-Flow procedure, *Monte Carlo* Scheme, on the 48 packages of Table 7.

instances group name	*Time* (s)	*Gap* (%)
j301	5.81	0.00
j302	6.33	0.00
j303	6.30	0.00
j304	6.04	0.00
j305	5.80	2.69
j3011	7.01	1.02
j30I2	8.01	0.00
j3013	5.71	6.43
j3014	6.71	3.50
j3015	7.37	0.52
j3021	5.23	0.33
j3022	6.35	0.00
j3023	6.49	0.00
j3024	6.90	0.00
j3025	5.26	4.06
j3031	7.12	0.93
j3032	7.91	0.00
j3033	5.23	0.00
j3034	5.24	0.00
j3035	5.60	0.00
j3041	4.75	1.91
j3042	6.02	1.02
j3043	6.36	0.83
j3044	7.25	0.00
j3045	4.52	2.03

REFERENCES

[1] B. Abbasi, S. Shadrokh and J. Arkat, Bi-objective resource-constrained project scheduling with robustness and makespan criteria. *Appl. Math. Comput.* **180** (2006) 146–152.
[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin and M.R. Reddy, Applications of network optimization, in *Network Models (Chapter 1). Handbooks Oper. Res. Manage. Sci.* **7** (1995) 1–83.
[3] R.V. Ahuja, T.L. Magnanti and J.B. Orlin. *Network flows: theory, algorithms and applications*. Prentice hall, Englewood Cliffs, N.J (1993).
[4] J. Alcaraz and C. Maroto, A robust genetic algorithm for resource allocation in project scheduling. *Ann. Oper. Res.* **102** (2001) 83–109.
[5] J. Alcaraz, C. Maroto and R. Ruiz, Improving the performance of genetic algorithms for the RCPSP problem, in *Proc. 9th Int. workshop on project management and scheduling* (2004) 40–43.
[6] C. Artigues and F. Roubellat, A polynomial activity insertion algorithm in a multiresource schedule with cumulative constraints and multiple nodes. *EJOR* **127** (2000) 297–316.
[7] C. Artigues, P. Michelon and S. Reusser, Insertion techniques for static and dynamic resource constrained project scheduling. *EJOR* **149** (2003) 249–267.
[8] C. Artigues and C. Briand, The resource-constrained activity insertion problem with minimum and maximum time lags. *J. Schedul.* **12** (2009) 447–460.
[9] K.R. Baker, *Introduction to sequencing and scheduling*. Wiley, N.Y (1974).

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

[10] P. Baptiste, *Resource constraints for preemptive and non preemptive scheduling*, MSC Thesis, University PARIS VI (1995).
[11] P. Baptiste and S. Demassey, Tight LP-bounds for resource constrained project scheduling. *OR Spectrum* **26** (2004) 251–262.
[12] P. Baptiste, P. Laborie, C. Lepape and W. Nuijten, Constraint-based scheduling and planning, in *Handbook of Constraint Programming* **22**, edited by F. Rossi, P. Van Beek. Elsevier (2006) 759–798.
[13] T. Baar, P. Brucker and S. Knust, Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem, in Meta-heuristics: Advances and trends in local search paradigms for optimization, edited by S. Voss, S. Martello, I. Osman and C. Roucairol. Kluwer Academic Publishers (1998) 1–8.
[14] C. Berge, *Graphes et Hypergraphes*. Dunod Ed (1975).
[15] J. Blazewiecz, K.H. Ecker, G. Schmlidt and J. Weglarcz, *Scheduling in computer and manufacturing systems*. 2th edn, Springer-Verlag, Berlin (1993).
[16] K. Bouleimen and H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *EJOR* **149** (2003) 268–281.
[17] C. Briand, A new any-order schedule generation scheme for resource-constrained project scheduling. *RAIRO - Oper. Res.* (2009) 297–308.
[18] P. Brucker and S. Knust, A linear programming and constraint propagation based lower bound for the RCPSP. *EJOR* **127** (2000) 355–362.
[19] P. Brucker, S. Knust, A. Schoo and O. Thiele, A branch and bound algorithm for the resource constrained project scheduling problem. *EJOR* **107** (1998) 272–288.
[20] P. Brucker, A. Drexl, R. Mohring, K. Neumann and E. Pesch, Resource-constrained project scheduling: notation, classification, models and methods. *EJOR* **112** (1999) 3–41.
[21] J. Carlier and P. Chrétienne. *Problèmes d'ordonnancements : modélisation, complexité et algorithmes*. Masson Ed, Paris (1988).
[22] J. Carlier and E. Neron, Computing redundant resources for the resource constrained project scheduling problem. *EJOR* **176** (2007) 1452–1463.
[23] J. Carlier and E. Neron, On linear lower bounds for the resource constrained project scheduling problem. *EJOR* **149** (2003) 314–324.
[24] H. Chtourou and M. Haouari, A two-stage-priority rule based algorithm for robust resource-constrained project scheduling. *Comput. Indust. Engin.* **12** (2008).
[25] N. Chistophides and C.A. Whitlock, Network synthesis with connectivity constraint: a survey. *Oper. Res.* (1981) 705–723.
[26] J. Coelho and L. Tavares, Comparative analysis of metaheuricstics for the resource constrained project scheduling problem. *Technical report*, Department of Civil Engineering, Instituto Superior Tecnico, Portugal (2003).
[27] G. Dahl and M. Stoer, A polyhedral approach to multicommodity survivable network design. *Numerische Math.* **68** (1994) 149–167.
[28] J. Damay, *Techniques de resolution basées sur la programmation linéaire pour l'ordonnancement de projet*. PH.D. Thesis, Université de CLERMONT-FERRAND (2005).
[29] J. Damay, A. Quilliot and E. Sanlaville, Linear programming based algorithms for preemptive and non preemptive RCPSP. *EJOR* **182** 1012–1022. (2007).
[30] S. Demassey, C. Artigue and P. Michelon, Constraint–propagation-based cutting planes: an application to the resource-constrained-project-scheduling problem. *INFORMS J. Comput.* **17** (2005) 1.
[31] E. Demeulemeester and W. Herroelen, New benchmark results for the multiple RCPSP. *Manage. Sci.* **43** (1997) 1485–1492.
[32] B. De Reyck and W. Herroelen, A branch and bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *EJOR* **111** (1998) 152–174.
[33] K. Djellab, Scheduling preemptive jobs with precedence constraints on parallel machines. *EJOR* **117** (1999) 355-367.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

[34] D. Dolev and M.K. Warmuth, Scheduling DAGs of bounded heights. *J. Algor.* **5** (1984) 48–59.
[35] D. Debels, B. De Reyck, R. Leus, M. Vanhoucke, A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *EJOR* **169** (2006) 638–653.
[36] B. Dushnik and W. Miller, Partially ordered sets. *Amer. J. Math.* **63** (1941) 600–610.
[37] P. Fortemps and M. Hapke, On the disjunctive graph for project scheduling. *Foundat. Comput. Decis. Sci.* **22** (1997) 195–209.
[38] D.R. Fulkerson and J.R. Gross, Incidence matrices and interval graphs. *Pac. J. Maths* **15** (1965) 835–855.
[39] R.L. Grahamson, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnoy-Khan, Optimization and approximation in deterministic scheduling: a survey. *Annal. Disc. Math.* **5** (1979) 287–326.
[40] S. Hartmann and D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem. *Eur. J. Operat. Res.* **207** (2010) 1–14.
[41] W. Herroelen, E. Demeulemeester and B. de Reyck, A classification scheme for project scheduling, in *Project Scheduling: recent models, algorithms and applications*. Kluwer Acad Publishers (1999) 1–26.
[42] W. Herroelen, Project Scheduling-Theory and Practice. *Prod. Oper. Manag.* **14** (2006) 413–432.
[43] M. Haouari, A. Gharbi, A improved max-flow based lower bound for minimizing maximum lateness on identical parallele machines. *Operat. Res. Lett.* **31** (2003) 49–52 .
[44] J. Josefowska, M. Mika, R. Rozycki, G. Waligora, J. Weglarcz, An almost optimal heuristic for preemptive Cmax scheduling of dependant task on parallel identical machines. *Annal. OR* **129** (2004) 205–216.
[45] R. Kolisch and A. Drexl, Adaptive search for solving hard project scheduling problems. *Naval Res. Logist.* **43** (1996) 23–40.
[46] R. Kolisch and S. Hartmann, Experimental investigation of heuristics for the resource constrained scheduling problem: an update. *EJOR* **174** (2006) 23–37.
[47] A. Kimms, Mathematical programming and financial objectives for scheduling projects. *Oper. Res. Manag. Sci.* Kluwer Academic Publisher (2001).
[48] R. Kolisch, A. Sprecher and A. Drexel, Characterization and generation of a general class of resource constrained project scheduling problems. *Manag. Sci.* **41** (1995) 1693–1703.
[49] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *Eur. J. Oper. Res.* **90** (1996) 320–333.
[50] R. Kolisch, R. Padman. An integrated survey of deterministic project scheduling. *Omega* **48** (1999) 249–272.
[51] R. Kolisch and S. Hartmann, Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis, in edited by J. Weglarcz. *Project Scheduling: recent models, algorithms and applications*, Kluwer Acad Press (1999).
[52] Y. Kochetov and A. Stolyar, Evolutionnary local search with variable neighbourhood for the resource constrained scheduling problem, in *Proc. 3 th int. Conf. Computer Sciences and Information Technologies, Russia* (2003).
[53] E.L. Lawler, K.J. Lenstra, A.H.G. Rinnoy-Kan and D.B. Schmoys, Sequencing and scheduling: algorithms and complexity, in *Handbook of Operation Research and Management Sciences, Vol 4: Logistics of Production and Inventory*, edited by S. C. GRAVES, A.H.G. Rinnoy-kan, P.H. Zipkin. North-Holland (1993) 445–522.
[54] R. Leus and W. Herroelen, Stability and resource allocation in project planning. IIE transactions. **36** (2004) 1–16.
[55] V.J. Leon and B. Ramamoorthy, Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *OR Spektrum* **17** (1995) 173–182.
[56] A. Mingozzi, V. Maniezzo, S. Ricciardelli and L. Bianco, An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Manage. Sci.* **44** (1998) 714–729.

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press

[57] M. Minoux, Network synthesis and optimum network design problems: models, solution methods and application. *Networks* **19** (1989) 313–360.
[58] P.B. Mirchandani and R.L. Francis, Discrete location theory. John Wiley and sons (1990).
[59] R.H. Mohring and F.J. Rademacher, Scheduling problems with resource duration interactions. *Methods Oper. Res.* **48** (1984) 423–452.
[60] A. Moukrim and A. Quilliot, Optimal preemptive scheduling on a fixed number of identical parallel machines. *Oper. Res. Lett.* **33** (2005) 143–151.
[61] A. Moukrim and A. Quilliot, A relation between multiprocessor scheduling and linear programming. *Order* **14** (1997) 269–278.
[62] R.R. Muntz and E.G. Coffman, Preemptive scheduling of real time tasks on multiprocessor systems. *J. ACM* **17** (1970) 324–338.
[63] K. Neumann, C. Schwindt and J. Zimmermann, *Project scheduling with time windows and scarce resources*. Springer, Berlin (2003).
[64] K. Nonobe and T. Ibaraki, *Formulation and tabu search algorithm for the resource constrained project scheduling problem.* In C.C. Ribeiro and P. Hansen, editors, Essays and surveys in metaheuristics. Kluwer Academic Publishers, Dordrecht (2002) 557–588.
[65] M. Palpant, C. Artigues and P. Michelon, LSSPER: solving the resource-constrained project scheduling problem with large neighbourhood search. *Ann. Oper. Res.* **131** (2004) 237–257.
[66] C.H. Papadimitriou and M. Yannanakis, Scheduling interval ordered tasks. *SIAM J. Comput.* **8** (1979) 405–409.
[67] P.M. Pardalos and D.Z. Du, *Network design: connectivity and facility location*. DIMACS Series 40, N.Y, American Math Society (1998).
[68] J.H. Patterson, A comparizon of exact approaches for solving the multiple constrained resource project scheduling problem. *Manag. Sci.* **30** (1984) 854–867.
[69] N. Sauer and M.G. Stone, Rational preemptive scheduling. *Order* **4** (1987) 195–206.
[70] N. Sauer and M.G. Stone, Preemptive scheduling of interval orders is polynomial. *Order* **5** (1989) 345–348.
[71] A. Schirmer, Case-based reasoning and improved adaptive search for project scheduling. *Naval Res. Logist.* **47** (2000) 201–222.
[72] S.S. Liu and C.J. Wang, Resource-constrained construction project scheduling model for profit maximization considering cash flow. *Automat. Constr.* **17** (2008) 966–974.
[73] P. Tormos and A. Lova, Project scheduling with time varying resource constraints. *Int. J. Prod. Res.* **38** (2000) 3937–3952.
[74] P. Tormos and A. Lova, A competitive heuristic solution technique for resource-constrained project scheduling. *Ann. Oper. Res.* **102** (2001) 65–81.
[75] P. Tormos and A. Lova, An efficient multi-pass heuristic for project scheduling with constrained resources. *Int. J. Prod. Res.* **41** (2003) 1071–1086.
[76] P. Tormos and A. Lova, Integrating heuristics for resource constrained project scheduling: One step forward. *Technical report*, Department of Statistics and Operations Research, University of Valencia (2003).
[77] V. Valls, F. Ballestin and S. Quintanilla, A hybrid genetic algorithm for the RCPSP. *Technical report*, Department of Statistics and Operations Research, University of Valencia (2003).
[78] V. Valls, B. Ballestin and S. Quintanilla, Justification and RCPSP: a technique that pays. *EJOR* **165** (2005) 375–386.
[79] P. Van Hentenryk, *Constraint Programming*. North Holland (1997).

https://doi.org/10.1051/ro/2012021 Published online by Cambridge University Press