Proceedings of the Federated Conference on Computer Science and Information Systems pp. 439–446

ISBN 978-83-60810-51-4

Flow Models for Project Scheduling with Transfer Delays

Alain Quilliot
LIMOS CNRS UMR 6158
Université Blaise Pascal
Bat ISIMA, BP 10125
Campus des Cézaux,
63173 Aubière, France
Email: alain.quilliot@isima.fr

Hélène Toussaint
LIMOS CNRS UMR 6158
Université Blaise Pascal
Bat ISIMA, BP 10125
Campus des Cézaux,
63173 Aubière, France
Email: toussain@isima.fr

***Abstract*—This paper deals with an extension of the *Resource Constrained Project Scheduling Problem* (RCPSP), which involves resource transfer delays. A flow model is used in order to formalize this extended RCPSP, which contains the standard RCPS, and leads us to introduce the *Timed Flow Polyhedron* and to state several structural results. This framework gives rise to a generic Insertion operator, as well as greedy/local search algorithms. We end with numerical tests. Introduction**

I. Introduction

DEALING with *Resource Constrained Project Scheduling Problems* (RCPSP: see [1]–[3]) means scheduling a set of tasks, submitted to temporal and resource constraints, while minimizing the induced Makespan value. This problem has been extensively studied: [4]–[7]; its theoretical analysis requires the use of sophisticated mathematical tools: linear programming, posets, hypergraphs...: [8]. While Standard RCPSP only involves deterministic non pre-emptive tasks and renewable resources, extended models address pre-emption: [9], time lags: [10], non renewable resources: [11], [12], non constant profiles, robustness: [13], deadlines and penalties, redundant resources…: [14]–[16]. A survey about RCPSP variants is available in [17].

RCPSP problems are usually NP-Complete, and getting exact results becomes hard as soon as there are more than 60 tasks and 4 resources: [18],[19]. Characterizing benchmark computational complexity is also difficult: [20]. Exact methods are most often branch and bound, cut generation and constraint propagation based: [12], [21]–[24]. Powerful lower bounds derive from column generation techniques applied to specific LP models, energetic reasoning processes or largest paths computing: [25], [26]. But efficient heuristics may be designed: greedy algorithms based on priority rules or insertion techniques: [13], [27], [28] local search methods: [29]–[31]. Dynamic RCPSP is most often handled through priority rule based algorithms: [28].

This paper studies an extension of the RCPSP Problem which involves *Resource Transfer Delays*: RCPSTDP: resources are transmitted from one task to another, and those communication tasks involve delays. Such a RCPSPTDP model corresponds to the case when every task takes place on a given production unit and when part of the resources (workers, equipments…) need to be transported from one unit to another. Coping with such a problem requires the existence of an explicit representation of the way resources transit from one production unit to another and, thus, leads us to make appear a *Network Flow* component into our model.

Network Flow Theory: [32], [33], is devoted to the handling of problems which involve the circulation of goods, people, energy, information.... It has been essentially used in order to model transportation, telecommunication and energy distribution systems: [32]. The existence of a link between the RCPSP and Network Flow Theory has already been noticed in [27], [28], [6], [14], and been used in order to get ILP formulations, as well as some specific insertion algorithms. Still, few works have explicitly involved the network flow machinery into the design of generic algorithms. So, we are first going to explain the way RCPSTDP may be cast into the Network Flow framework, while introducing the *Flow Polyhedron Vertex Subset* related to a RCPSTDP instance. Next, we shall state several structural results about connectivity and cut management. Finally, we shall derive from this theoretical work a generic *insertion* mechanism, close to the insertion mechanisms which were proposed in [27], [28] and use it in order to design and test greedy and local search algorithms.

II. Network Flow Model Related to a RCPSTDP Instance

A. Preliminary Notations and Definitions

We denote by ← the value allocation operator: "$x \leftarrow a$" means that variable x takes value a. $\boldsymbol{Q}$ is the rational number set. If τ is some partial order relation, then $\tau^{=}$ is the relation (τ or =) and $\mathrm{Tr}(\tau)$ is the *transitive closure* of τ. An oriented graph (network) G with node set Z and arc set E is denoted by $G = (Z, E)$. An arc e with origin/destination nodes x and y

is denoted by (x, y). A partial graph (sub-graph) of G is the restriction of G to some subset of E (Z).

B. The Non Preemptive RCPSTDP Problem

A Standard RCPSP instance I = (V, K, R, r, d, <<) is defined by:

- a set V of non pre-emptive tasks: every task v in V is endowed with some duration $d_v > 0$ and must be run as a whole during some time window with length d_v;
- a binary no circuit precedence relation <<, which is defined on the set V: v << 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, task v requires a $r_{k,v}$ amount of resource k to be available, and forbids any other task to use it. Once it is over, task v gives this resource back;

Solving I means computing, for any $v \in V$, its starting time $T_v \geq 0$, in such a way that:

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

An instance $I\text{-}TD$ = (V, K, R, r, d, <<, $\mathcal{D}$-$\mathcal{L}ag$, Depot) of the Resource Constrained Project Scheduling with Transfer Delays Problem (RCPSTDP) is defined as above, while taking into account the Delay function $\mathcal{D}$-$\mathcal{L}ag$ whose meaning is:

- tasks of V are run at different places inside some "production" space, and resources circulate between these places. At time 0, resources are all located at a same place Depot, and they must be back to Depot for the project to be over. Then the Delay **Q**-valued function D-Lag, associates, with any pair v, w in V $\cup$ {Depot}, a value D-lag(v, w) $\geq$ 0: if task v (or Depot) transmits some resource to task w (or to Depot) or if v << w (v transmits some output to w which uses it as an input), then transferring this resource requires a D-lag(v, w) delay between the ending time of v (0 if v = Depot) and the starting time of w. (Delay Constraint)

So, solving $I\text{-}TD$ means simultaneously computing a Time vector T as in the simple case, and an ad hoc description F of the way resources are provided to the tasks.

Remark 1. RCPSTD and RCPSP with Time Lags are quite different problems, since delays $\mathcal{D}$-$\mathcal{L}ag$(v, w) only impact tasks v, w which exchange resources.

C. Linking Network Flows with RCPSTDP: Timed Flows

We may now explain what may be an "ad hoc description F of the way resources are provided to the tasks", and get a formal framework for the RCPSPTD Problem;

Recall: Network 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; a **Q**-valued E-indexed vector f is a ϕ-flow vector iff:

$$\forall z \in Z, \sum_{z\_origin\_of\_e} f_e = \sum_{z\_destination\_of\_e} f_e$$

(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 **Q**-flow vector any collection $f = (f(i), i \in I)$, where every f(i), $i \in I$, is a $\phi(i)$-flow vector.

The Activity Network. Let $I\text{-}TD$ = (V, K, R, r, d, <<, $\mathcal{D}$-$\mathcal{L}ag$, Depot) be a RCPSPTD instance. We derive from $I\text{-}TD$ the Activity Network $\mathcal{N}(V) = (V^*, E^*)$ by introducing two auxiliary tasks Start and End, and by setting:

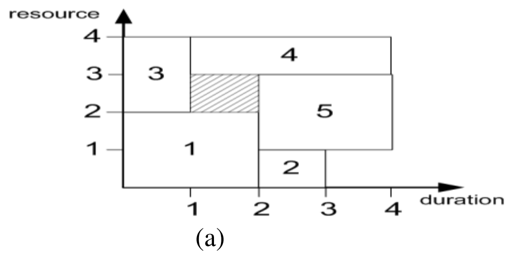
- $V^* = V \cup \{\text{Start, End}\}$ = node set of $\mathcal{N}(V)$;
- $E^* = \{(v, v'), v, v' \in V\} \cup \{(\text{Start}, v), v \in V \cup \{\text{End}\}\} \cup \{(v, \text{End}), v \in V \cup \{\text{Start}\}\}$ = arc set of $\mathcal{N}(V)$.

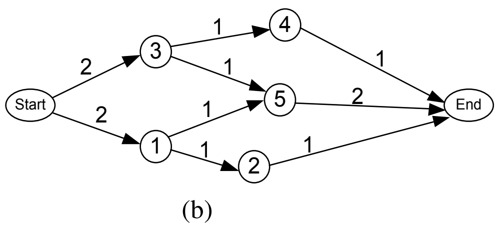
We define the E*-indexed length vector d^* by setting:

- for any $v \in V$, $d^*_{(\text{Start}, v)} = \mathcal{D}\text{-}\mathcal{L}ag(\text{Depot}, v)$ and $d^*_{(v, \text{End})} = d_v + \mathcal{D}\text{-}\mathcal{L}ag(v, \text{Depot})$
- $d^*_{(\text{End, Start})} = -\infty$;
- for any $v \in V$, $w \in V \cup \{\text{End}\}$, $d^*_{(v, w)} = d_v + \mathcal{D}\text{-}\mathcal{L}ag(v, w)$.

We provide the node set V^* with 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 $E^*_{<<}$ by setting: $E^*_{<<} = \{(v, v'), v, v' \in V$ such that $v\ \mathcal{T}r(<<)\ v'\} \cup \{(\text{Start}, v), (v, \text{End}), v \in V\}$, where $\mathcal{T}r(<<)$ is the transitive closure of the << relation.

Feasible solutions of $I\text{-}TD$ and r^*-flow vectors. Then, an explicit representation of the circulation of resources between the tasks of V, consists into a r^*-flow vector $F = (F(k), k \in K)$, defined on the Activity Network $\mathcal{N}(V)$, which may be viewed as transporting the resources $k \in K$, from Depot (the source-node Start) to Depot (the end-node End), while providing the activities $v \in V$ with the required resources. We define the support arc subset E(F,<<) of F by setting: $E(F, <<) = E^*_{<<} \cup \{(v, w), v, w \in V$ such that $F_{(v, w)}$ is non null$\}$. Clearly, E(F,<<) has no circuit, and so we say that F is no circuit. As a matter of fact, we easily see (see for instance [27, 28]), that even if we only deal with a simple RCPSP instance I = (V, K, R, r, d, <<, $\mathcal{D}$-$\mathcal{L}ag$), we may derive, as described in Figure 1, such a flow F from any feasible schedule T.





(b)

Fig. 1 (a) Gantt chart – (b) Flow representation

Casting RCPSPTDP into a formal framework: Timed Flows. Let F be a no circuit r^*-flow vector defined on the Activity Network $\mathcal{N}(V)$ and T be a time V-indexed vector such that the pair (F, T) defines a feasible solution of $\mathcal{I}$-$\mathcal{TD}$. If we extend the time vector T to $V \cup \{Start, End\}$ by setting:

- $T_{Start} = 0$; $T_{End} = \Delta = Makespan(T) = Sup_{v \in V} (T_v + d_v + \mathcal{D}\text{-}\mathcal{L}ag(v, Depot))$;

then, for any arc e = (v, w) in the arc set E^* of $\mathcal{N}(V)$, we get the implication:

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

This leads us to define a Timed (r^*, d^*)-Flow as being any such pair (F, T) made of a no circuit r^*-flow vector F and a time vector T such that (P1) is true. One easily checks that any such a Timed (r^*, d^*)-Flow (F, T) defines a feasible solution of $\mathcal{I}$-$\mathcal{TD}$. This notion of Timed Flow allows us to reformulate RCPSTDP as follows:

RCPSPTD Timed Flow Reformulation: solving the RCPSTDP instance $\mathcal{I}$-$\mathcal{TD}$ = (V, K, R, r, d,<< , $\mathcal{D}$-$\mathcal{L}ag$, Depot) means computing, on the Activity network $\mathcal{N}(V)$, a Timed (r^*, d^*)-Flow (F, T) such that T_{End} is the smallest possible.

Also, following ([27, 28]) one easily checks that:

Theorem 1: Standard RCPSP Reformulation Theorem
Any feasible solution T of the Standard RCPSP instance $\mathcal{I}$ = (V, K, R, r, d,<<) may be extended into a feasible solution (F, T) of the RCPSTDP instance $\mathcal{I}$-$\mathcal{TD}$ = (V, K, R, r, d, <<, *0*, Depot).

D. A Connectivity Theorem

Part of the efficiency of the flow machinery derives from properties of the flow polyhedron. Thus, one may ask about the part of this polyhedron made with the no circuit r^*-flow vectors.

The No Circuit r^*-Flow Polyhedral Vertex Set: the set of all r^*-flow vectors $F \geq 0$ defined on the Activity Network $\mathcal{N}(V)$ defines a bounded polyhedron $\mathcal{P}_{r^*}$. A r^*-flow vector F is a vertex of $\mathcal{P}_{r^*}$ which contains no non null alternated cycle, that means there is no cycle ($v_0, v_1, ..., v_n = v_0$) such that:

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

We denote by $\mathcal{S}_{r^*}$ the vertex set of this polyhedron. This vertex set is endowed with a canonical adjacency relation $\mathcal{R}$, which may be characterized as follows:

- let Γ be some even cycle ($v_0, v_1, ..., v_n = v_0$) in $\mathcal{N}(V)$: the alternated cycle flow f^Γ is defined by:
  - $f^\Gamma_e = +1$ for any arc e = $(v_0, v_1), (v_2, v_3), ..., (v_{n-2}, v_{n-1})$;
  - $f^\Gamma_e = -1$ for any arc e = $(v_2, v_1), (v_4, v_3), ..., (v_0, v_{n-1})$.
- F, F' in $\mathcal{S}_{r^*}$ are $\mathcal{R}$-adjacent if there exists some resource $k_0 \in K$, some even cycle Γ and some number $\lambda \geq 0$, 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, value λ is unique, and F' derives from F through redirection of F(k) on Γ.

It comes from LP Theory that $\mathcal{S}_{r^*}$ is connected for the relation $\mathcal{R}$. Since we deal here with no circuit r^*-flows, we are led to ask whether restricting the Redirection scheme to the set $\mathcal{SN}_{r^*}$ of those specific vertices of $\mathcal{P}_{r^*}$ which define no circuit r^*-flow vectors maintains this connectivity property. We call this set $\mathcal{SN}_{r^*}$ the No Circuit r^*-Flow Polyhedral Vertex Set. Then we may state:

Theorem 2: Connectivity Theorem
If we suppose that, for any $k \in K$, v, $w \in V$, we have: $r_{k,v} + r_{k,w} \leq R_k$ (Parallelism Hypothesis), then the No Circuit r^*-Flow Polyhedral Vertex Set $\mathcal{SN}_{r^*}$ is connected for the canonical adjacency relation $\mathcal{R}$.

Comment: this means that one may handle timed flows (and RCPSP instances) through classical local search procedures (cancelling cycle procedures…).

Proof of theorem 2 (Sketch of the Proof)
Let us first define a linear r^*-flow vector as being a no circuit r^*-flow vector $F \geq 0$ which is such that the transitive extension of the support arc set E(F,<<) is linear. So we denote by $\mathcal{SNL}_{r^*}$ the subset of $\mathcal{SN}_{r^*}$ which is made with linear r^*-flow vectors. If σ is some linear ordering of $V \cup \{Start, End\}$, which is compatible with <<, we denote by $\mathcal{SN}_{r^*}(\sigma)$ the subset of $\mathcal{SN}_{r^*}$ which corresponds to the case when σ may be viewed as a linear extension of the transitive extension of E(F, <<). Then we check, by using ad hoc flow redirection processes involving cycles with length 4 that:

Lemma. $\mathcal{SN}_{r^*}(\sigma)$ is connected for the $\mathcal{R}$ relation. Also, if we suppose that, for any $v \in V$, $k \in K$, $r_{k,v} \neq 0$ *and that the* parallelism holds, then we may state that:

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

Clearly, this lemma allows us to conclude to the $\mathcal{R}$-connectivity of $\mathcal{SN}_{r^*}$ 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 $\mathcal{S}_{r^*}^{\delta}$ and $\mathcal{SN}_{r^*}^{\delta}$ the respective related polyhedron vertex sets and by $\mathcal{R}^{\delta}$ the related adjacency relation. It comes from above that $\mathcal{SN}_{r^*}^{\delta}$ is connected for the relation $\mathcal{R}^{\delta}$. Also, we see that if F is some vertex in $\mathcal{SN}_{r^*}$, then the r*-flow vector F^{δ} defined by:

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

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

So we conclude by checking that any pair F, H of elements of $\mathcal{SN}_{r^*}$, may be connected by a path Γ which is the limit, when δ converges to 0, of some path sequence Γ^{δ}, $\delta > 0$, where every path Γ^{δ} connects F^{δ} and H^{δ} in $\mathcal{SN}_{r^*}^{\delta}$. End-Proof.

Remark: one easily check that the Parallelism hypothesis cannot be removed.

III. Insertion scheme, insertion problem and algorithms

As told in 2, the main motivation for introducing the Timed Flow formalism is provided by the prospect of applying ad hoc network flow algorithmic tools to RCPSTDP instances. So the current Section III is devoted to the description of the way it can be done. Basically, our RCPSTDP algorithms work while performing insertion/removal processes which may be compared with those which have been proposed in [27, 28] for standard RCPSP: 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 given Cut of the currently inserted task set: 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 II. More precisely, at any time during the process of some RCPSTDP instance ***I-TD*** = (V, K, R, r, d, $\ll$, *D-Lag*, Depot), we are provided with some Inserted Activity subset W of V, with a no circuit r^*- flow vector F defined on the Activity Network $\mathcal{N}(W)$, and with two positive (or null) $\boldsymbol{Q}$-valued time vectors T and T^*, both with indexation on W^*, in such a way that, for any v in W^*:

(P4)

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

Clearly, the pair (F, T) defines a timed (r^*, d^*)-flow on $\mathcal{N}(W)$. Then, performing an Insertion means picking up some task v_0 which is not in W, computing some Cut, i.e. a partition of W into two subsets U and W−U, such that no flow amount goes from (W−U) $\cup$ {End} to U $\cup$ {Start}, and turning (F, T), through the resolution of the Insertion-Flow Problem, into a convenient timed (r^*, d^*)-flow defined on $\mathcal{N}(W \cup \{v_0\})$, in such a way that 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 a more accurate way, we need to introduce the Flow-Insertion Problem. Meanwhile, we may illustrate the general insertion/removal mechanism through the drawings of Figure 1, which represent, (a), a partial solution with 4 tasks and a Cut (U = {1, 3}, W−U = {2, 4}), and, (b), the insertion of task $v_0 = 5$ (with $d_5 = 2$ and $r_5 = 2$) into this Cut.

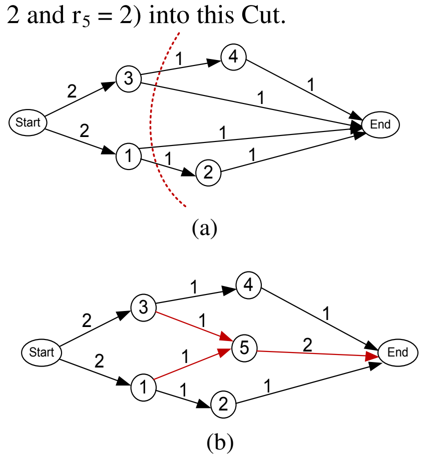


Fig. 2 (a) The flow and the "cut" (dotted line) - (b) the resulting flow after insertion

A. The Insertion Flow Problem

The above considerations lead us to formalize the Insertion-Flow Problem related to a given Cut as follows: we say that an oriented graph $\mathcal{N} = (X, E)$ is almost-bipartite, if there exists some node z_0 in X such that the restriction of $\mathcal{N}$ 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 A-indexed vectors Π, *Out*, with two positive (or null) Q-valued B-indexed vectors Π^*, *In*, with some Q-valued positive vectors Δ with indexation on $(A \cup \{z_0\}).(B \cup \{z_0\})$, and with a positive (or null) coefficient ρ, such that: $\Sigma_{x \in A}\, Out(x) = \Sigma$

$_{y\in B}$ $In(y) \geq \rho$; then we say that a vector $G = (G_{x,y} \geq 0, x \in A \cup \{z_0\}, B \cup \{z_0\}) \geq 0$, is an Insertion-Flow vector related to those data: (X, E), z_0, A, B, *In*, *Out*, Π, Π^*, Δ, ρ) iff:

- for any x in A, $Out_x = \Sigma_{y\in B \cup \{x0\}} G_{x,y}$;
- for any y in B, $In_y = \Sigma_{x\in A \cup \{x0\}} G_{x,y}$;
- $\rho = \Sigma_{y\in B} G_{z0,y} = \Sigma_{x\in A} G_{x,z0}$.

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

- Make1(G) = $\text{Sup}_{x \in A, y\in B \text{ such that } (x, y) \in E \text{ or } Gx,y \neq 0} (\Pi_x + \Pi^*_y + \Delta_{x,y})$;
- Make2(G) = $\text{Sup}_{x \in A, y \in B \text{ such that } ((x, z0) \in E \text{ or } Gx,z0 \neq 0) \text{ and } (z0, y) \in E \text{ or } Gz0,y \neq 0)} (\Pi_x + \Pi^*_y + \Delta_{x,z0} + \Delta_{z0,y})$;
- I-Makespan(G) = Sup(Make1(G), Make2(G)).

This definition leads us to introduce the following **Insertion-Flow Problem**: {Given (X, E), z_0, A, B, *In*, *Out*, Π Π^*, Δ, ρ, 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 situation which prevails during the previously described insertion process, and if we suppose that Card(K) = 1, then we clearly see that we should think: (P5)

- A = U∪{Start}; B = (W – U)∪{End}; E = (v, w), v∈A, w∈B, such that v *Tr*(≪) w;
- $z_0 = v_0$; $\rho = r_{k,v0}$;
- for any v in U ∪ {Start}, $\Pi_v = T_v$ and $Out_v = \Sigma_{w\in B} F(k)_{(v,w)}$;
- for any w in B = (W – U) ∪ {End}, $\Pi^*_w = T^*_w$ and $In_w = \Sigma_{v\in A} F(k)_{(v,w)}$.
- for any pair v, w in $(A \cup z_0).(B \cup z_0)$, $\Delta_{v,w} = d^*_{(v,w)}$.

In order to deal with this Insertion-Flow problem, we design an Insertion-Flow procedure which:

- first computes the set $\Lambda_{E,\ z0}$ of the possible attachment values: for a given Insertion-Flow vector G, its attachment value u(G) will be given by:
  u(G)) = Sup $(\Pi_x + \Delta_{x,z0})$, x∈ A such that $(G_{x,\ z0} \neq 0$ or $(x, z_0) \in E)$;
- next, **For** any u in $\Lambda_{E,\ z0}$, computes some Insertion-Flow G such that u = u(G), and performs the following loop:
  - **While** Possible do: Make decrease, through the search of some Redirection Path, the G value on the arcs (x, y) such that I-Makespan(G) = $G_{x,y}$, while making u(G) remain unmodified;
- ends while keeping with the best Insertion-Flow G which was computed.

Theorem 3: Insertion-Flow Theorem
The Insertion-Flow Procedure solves in an exact way the Insertion-flow Problem in polynomial time.

Proof of theorem 3 (Sketch of the Proof)
The key point is that u(G) remains unmodified during the While Possible loop: then, a standard flow reasoning makes appear that if G is some Insertion-Flow vector, G(u) derives from the While Possible loop for a given u, and if we have both u(G) = u and I-Makespan(G(u)) > I-Makespan(G), then the cycle decomposition of the flow vector G – G(u) makes appear some Redirection Path Γ which make possible improving G(u). End-Proof.

Fast-Try-Insertion and Lex-Insertion-Flow Procedures: As a matter of fact, before insertion is effectively performed, it must be tried several times. In order to speed this trial process, we designed a greedy approximation Fast-Try-Insertion heuristics, which ties the node z_0 with the nodes x of A which are the most likely to make increase I-Makespan in case $G_{x,\ z0} = 0$, and proceed the same way in order to distribute flow values from A to B. By the same way we minimize the number of arcs (v, w) such that (v (Not Tr(<<)) w and $F_{(v,\ w)} \neq 0$) by turning Insertion-Flow into a Lex-Insertion-Flow procedure which tends to allocate G values to arcs of E, while E increases every time a new resource k is handled.

B. Generic Flow Algorithms for the RCPSTDP

A Greedy Insertion Algorithm *RCPSTDP-Greedy-Flow*
This algorithm works through successive insertions as explained at the beginning of Section III. Clearly, a key point is about the computation of the Cut U of the set W of the currently inserted tasks, prior to the call to the Insertion-Flow procedure. But 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: Cut Theorem
The search for a best Cut as defined above is NP-Complete.

Proof of theorem 4:
Let us consider the following input situation for the Best Cut Problem:

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

Determining whether the optimal value of this Best Cut instance is no more than 2 means solving some 2-Partition problem instance. End- Proof.

Still, if v is some node W ∪ {End}, we may set Cut(v) = {w ∈ W, $w \neq v$, such that $T_w \leq T_v$}. So, while searching for a best Cut U is difficult, we can easily scan the set W, and choose U = Cut(v) in such a way that an application of the Insertion-Flow Procedure yields the best possible Makespan value. The whole process, which may be easily randomized, may be summarized as follows:

```
RCPSP-Greedy-Flow Algorithm
Input: the instance I-TD = (V, K, R, r, d, <<, D-Lag, Depot) of the
RCPSTDP problem;
Output: a timed (r*, d*)-flow (F, T), and the related Makespan
value Δ
Init.: W ← Nil; Δ ← 0; F ← 0; T_Start = T_End = T*_Start = T*_End ← 0;
Main Loop: RCPSP-Flow ← Packet-Insertion(Nil);
```

Packet-Insertion Procedure

```
Input: the instance I-TD = (V, K, R, r, d, <<, D-Lag, Depot) of the RCPSTDP problem, a subset W of V, given together with a related no circuit $r^*$-flow vector F, and two related vectors T and $T^*$ as in (P4).
Output: a timed ($r^*$, $d^*$)-flow (F, T), and the related Makespan value $\Delta$
  Initialization: $S \leftarrow V - W$; $S_{Aux} \leftarrow S$;
While $S_{Aux} \neq Nil$ do
    Randomly Pick up $v_0$ in $S_{Aux}$ and Remove it from $S_{Aux}$;
    Compute $v_1$ in $W \cup \{End\}$, such that the application of the Fast-Try-Insertion-Flow procedure to:
        o $X = W \cup \{Start, End, v_0\}$; $A = Cut(v_1) \cup \{Start\}$; $B = X - A - \{v_0\}$;
        o $z_0$, In, Out, $\Pi$ $\Pi^*$, $\Delta$, $\rho$, $E = \{(x, y), x \in A \cup \{v_0\}, y \in B \cup \{v_0\}$, as in (P5);
    yields the best possible Makespan value;
    Let $u_1$ be the related attachment value: Apply Lex-Insertion-Flow to $Cut(v_1)$ and $u_1$, and perform the insertion of $v_0$ into the timed ($r^*$, $d^*$)-flow (F, T) in an effective way (update F, T and T* values);
    $W \leftarrow W \cup \{v_0\}$;
```

A local search RCPSTDP-LS-Flow Algorithm

The above Packet-Insertion operator gives rise in a generic way to a local search operator Transform-Insertion. 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, i.e applying some Reverse-Insertion procedure) and, next, come back to inserting the activities of S into the pair (F, T). Transform-Insertion operates on any timed (r^*, d^*)-flow (F, T), through a parameter $S \subseteq V$:

```
Transform-Insertion(S) Procedure:
Packet-Reverse-Insertion(S);
Packet-Insertion(V - S).
```

Provided with this operator, we may design several local search algorithms based upon Descent, Tabu or Simulated Annealing control, while picking up the parameter subset as the task subset defined by a critical path (Crit-Path strategy) or by the tasks which are simultaneously running at some critical time t (Antichain strategy).

IV. Numerical tests

We performed several experiments, on PC AMD opteron 2.1GHz, while using gcc 4.1 compiler. We tried several instance packages, most of them derived from the PSPLIB testbed, and, for every package, we tried both ***RCPSTDP-Greedy-Flow*** and ***RCPSTDP-LS-Flow***, while using replication techniques. Since we were not provided with optimal values for general RCPSTDP, we first tested the case *D-Lag* = 0, which means that RCPSP and RCPSPTD are the same problems, as well as ad hoc instances, with *D-Lag* designed in such a way that the optimal values of both problems were the same. For every instance, we kept memory of the number of activities N-Ac, the number of resources N-Res, the number of involved replications N-Re, and got an evaluation of:

- Time = CPU time in seconds for the N-rep replications;
- Gap-LB (%) = gap between our values and: in case of 30 job instances, the optimal value; in case of 60 and 120 job instances, the best lower bound value.
- Gap-TB (%) = gap between our values and: in case of 30 job instances, the optimal value; in case of 60 and 120 job instances, the trivial (largest path) lower bound value.

A. Experiments on PSPLIB instances with *D-Lag* = 0.

Since our models and algorithms are generic, they may be used in order to deal with standard RCPSP instances, and it is interesting to test their efficiency in such a specific context. The following tables 1 and 2 provide us with average results for the algorithms ***RCPSTDP-Greedy-Flow*** and ***RCPSP-LS-Flow***, related to the PSPLIB packages: 30 jobs, 60 jobs, 120 jobs, when *D-Lag* = 0. In this specific case, the Fast-Try-Insertion procedure is used, since it is designed in such a way that it ensures the same result as the full procedure when D-Lag = 0. We see that the induced results are very satisfactory, taken into account the simplicity and the generic features of those algorithms which derive from our Timed Flow framework. In case of 60 and 120 job instances, they provide us with some of the best available results.

TABLE I.
RCPSP-GREEDY-FLOW PROCEDURE, MEAN RESULTS

N-Ac	N-res	N-re	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	5000	31.6	0.53	0.53
30	4	50000	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

TABLE II.
RCPSP-LS-FLOW PROCEDURE, MEAN RESULTS

N-Ac	N-Re	Strategy	Gap-TB	Gap-LB	Time (s)
30	10	Crit-Path	0.85	0.85	3.77
30	10	Antichain	0.36	0.36	1.64
30	50	Crit-Path	0.49	0.49	19.41
30	50	Antichain	0.12	0.12	8.42
60	10	Crit-Path	14.34	4.97	20.11
60	10	Antichain	12.70	3.82	9.33

60	50	Crit-Path	13.69	4.4	101.50
60	50	Antichain	11.93	3.34	46.93
120	10	Crit-Path	45.41	15.80	210.80
120	10	Antichain	38.67	12.41	63.73
120	50	Crit-Path	43.71	14.45	1055.90

B. Experiments on PSPLIB instances with *D-Lag* such that RCPSP and RCPSTDP optimal values are the same.

This sequence of tests involves "difficults" instances, which are deduced from standard RCPSP instances in the following way: we start from a standard RCPSP instance **I** of the PSPLIB and from some optimal of almost optimal solution T of this instance; then we randomly generate *D-Lag* values in such a way that T remains a feasible schedule for the related RCPSTDP instance. Since, for every pair (v, w) of actions of **I**, which are not run in parallel according to T, it is possible to compute a maximal D-Lag value Max-Lag(T, v, w) which is compatible with T, we generate our D-Lag values with a mean ratio D-Lag/Max-Lag which vary from 10% to 50% (difficult instances). Then the following table 3, provides us with results related to such ad hoc PSPLIB instances with 30 jobs distributed into 5 groups of 480 instances, according to the value of the mean ratio D-Lag/Max-Lag, the number of replications being here always equal to 100.

TABLE III.
RCPSP-GREEDY-FLOW PROCEDURE, 100 REPLICATIONS, MEAN RESULTS, AD HOC PSPLIB INSTANCES

Group-Instance	Gap (%)	Time (s)
1 - 10%	2.50	2,71
2 – 20%	3.21	2,79
3 – 30%	7.35	2,92
4 – 40%	8.50	2,94
5 – 50%	14.50	3.12

Comment: Clearly, we see that the largest is the D-Lag/Max-Lag mean ratio, the most difficult are the instances. Still, we may consider that results remain satisfactory, Also, we notice that using the full Try-Insertion procedure has an impact on the speed of the process.

C. Experiments on Ad Hoc instances with *D-Lag* such that RCPSP and RCPSTDP optimal values are the same.

This last sequence of tests is related to ad hoc instances **I** with only one resource, which have been generated in such a way that the optimal value of **I** is exactly the quantity ($\Sigma_{v \text{ in } V}$ $r_v.d_v$)/R. Thus, the parallelism level (the mean number of actions which are simultaneously run) of those instances is high. Then we generate the D-Lag values and the << precedence relations in such a way that the ratio between the mean D–Lag values and the mean length of the actions of V vary from 10% to 50% (difficult instances).

The following figure 3 shows the skeleton of such an instance, i.e. the time/resource drawing which represents the schedule T: << precedence relations and D-lag values are generated in such a way that T remains feasible (and thus optimal). For instance, we may here generate (or not) the precedence relation 4 << 22 and a D-Lag(4, 22) value may be generated between 0 and Max-Lag(4, 22) = 7.

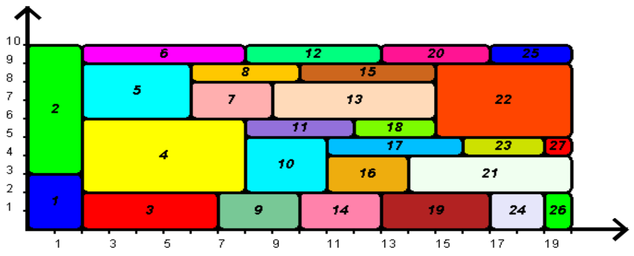


Fig. 3 The skeleton of an Ad Hoc 1 resource instance

Following Table 4 provides us with results for this kind of ad hoc instances, with N-Act between 22 and 37, such that RCPSP and RCPSTDP optimal values be the same, and *D-Lag* mean value vary from 10% to 50% of mean value of the task durations (difficult instances). The number of replications is once again equal to 100. One remarks that those specific instances are more difficult than the previous one, especially in the case of the last group. It makes appear that when the D-Lag values are similar to the duration of the tasks and when the parallelism level is high, greedy insertion is not enough in order to tackle the difficulty of the problem.

TABLE IV.
RCPSP-GREEDY-FLOW PROCEDURE, 100 REPLICATIONS, AD HOC INSTANCES WITH HIGH PARALLELISM LEVEL

Group-Instance	Gap (%)	Time (s)
1 - 10%	13.50	2.11
2 – 20%	13.00	2.11
3 – 30%	22.00	2.12
4 – 40%	18.50	2.32
5 – 50%	84.50	2.54

D. Fast-Try-Insertion versus Full Try-Insertion.

We test here the replacement of the Try-Insertion procedure of Section III.A, which involves path search, by a greedy Fast-Try-Insertion procedure, which involves simple swith moves. We consider here the same instances as in previous sub-section B, and then get the following results: We notice that this replacement does not lower in a significant way the quality of the solutions, and clearly speeds the process.

TABLE V.
RCPSP-GREEDY-FLOW PROCEDURE, 100 REPLICATIONS, MEAN RESULTS, AD HOC PSPLIB INSTANCES, FAST-TRY-INSERTION

Group-Instance	Gap (%)	Time (s)
1 - 10%	2.85	0.71
2 – 20%	3.80	0.79
3 – 30%	8.23	0.92
4 – 40%	9.80	0.94
5 – 50%	18.10	1.12

REFERENCES

[1] R. Kolisch, R. Padman. "Deterministic project scheduling", *Omega*, 48, pp. 249-272 (1999)

[2] P. Brucker, A. Drexl, R. Mohring, K. Neumann, E. Pesch. "Resource-constrained project scheduling: notation, classification, models and methods", *EJOR* 112, pp. 3-41 (1999)

[3] P. Baptiste, P. Laborie, C. Lepape, W. Nuijten. "Constraint-based scheduling/planning", in F.Rossi, P.Van Beek Eds, *Handbook Constraint Prog.*, Chap 22, pp.759-798, Elsevier, (2006)

[4] N. Sauer, M.G. Stone, "Rational preemptive scheduling", *Order* 4, pp. 195-206, (1987)

[5] J. Blazewiecz, K.H. Ecker, G. Schmlidt, J. Weglarcz. *Scheduling in computer and manufacturing systems*, 2-nd edn, Springer-Verlag, Berlin (1993)

[6] W. Herroelen. "Project Scheduling-Th./Pract"., *Prod./Op. Management*, 14, 4, pp. 413-432 (2006)

[7] S.S. Liu, C.J.Wang. "RCPSP profit max with cash flow", *Aut. Const.* 17, pp. 966-74 (2008)

[8] J. Damay, A. Quilliot, E. Sanlaville. "Linear programming based algorithms for preemptive and non preemptive RCPSP", *EJOR*, 182, 3, pp. 1012-1022 (2007)

[9] A. Moukrim, A. Quilliot. "Preemptive scheduling on parallel machines", *O.R Let.*, 33, pp. 143-51 (2005)

[10] B. De Reyck, W. Herroelen. "Branch/bound for the resource-constrained project scheduling problem with generalized precedence relations", *EJOR* 111, pp. 152-174 (1998)

[11] R.H. Mohring, F.J. Rademacher. "Scheduling problems with resource duration interactions", *Methods of Operat. Research* 48, pp. 423-452 (1984)

[12] A. Kimms. *Mathematical programming and financial objectives for scheduling projects*, Operations Research and Management Sciences, Kluwer Academic Publisher (2001)

[13] H. Chtourou, M. Haouari. "A two-stage-priority rule based algorithm for robust resource-constrained project scheduling", *Computers and Industrial Engineering*, 12 pages (2008)

[14] M. Haouari, A. Gharbi. "A improved max-flow based lower bound for minimizing maximum lateness on identical parallele machines", *OR Letters* 31, pp. 49-52 (2003)

[15] B. Abbasi, S. Shadrokh, J. Arkat. "Bi-objective resource-constrained project scheduling with robustness and makespan criteria", *App. Math. and Computation* 180, pp. 146-152 (2006)

[16] J. Carlier, E. Neron. "Computing redundant resources for the resource constrained project scheduling problem", *EJOR* 176, pp. 1452-1463 (2007)

[17] S. Hartmann, D. Briskorn. "A survey of variants of RCPSP". *EJOR*. 207:1-14, (2010)

[18] R. Kolisch, S. Hartmann. "Heuristic for RCPSP: classification and computational analysis", in J. Weglarcz Eds, *Project Scheduling: models and applications*, Kluwer Press, (1999)

[19] E. Demeulemeester, W. Herroelen. "New benchmark results for the RCPSP", *Management Science,* 43, pp. 1485-1492 (1997)

[20] R. Kolisch, A. Sprecher, A. Drexel. Characterization and generation of a general class of resource constrained project scheduling problems, *Management Science* 41, (10), pp.1693-1703 (1995)

[21] P. Brucker, S. Knust, A. Schoo, O. Thiele. "A branch and bound algorithm for the resource constrained project scheduling problem", *EJOR* 107, pp. 272-288 (1998)

[22] J. Josefowska, M. Mika, R. Rozycki, G. Waligora, J. Weglarcz. "An almost optimal heuristic for preemptive Cmax scheduling on parallel machines", *Annals of O.R.,* 129, pp. 205-16 (2004)

[23] A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco. "An exact algorithm for RCPSP based on a new mathematical formulation", *Management Science* 44, pp. 714-729 (1998)

[24] J.H. Patterson. "A comparizon of exact approaches for solving the multiple constrained resource project scheduling problem", *Management Sciences* 30 (7), pp. 854-867 (1984)

[25] P. Baptiste, S. Demassey. "Tight LP-bounds for RCPSP", *OR Spect.* 26, 2, pp. 251-62 (2004)

[26] P. Brucker, S. Knust. "A linear programming and constraint propagation based lower bound for the RCPSP", *EJOR* 127, pp. 355-362 (2000)

[27] C. Artigues, F. Roubellat. "A polynomial activity insertion algorithm in a multiresource schedule with cumulative constraints and multiple nodes", *EJOR* 127-2, pp. 297-316 (2000)

[28] C. Artigues, P. Michelon, S. Reusser. "Insertion for static/dyn. RCPSP", *EJOR* 149, pp. 249-267 (2003)

[29] K. Bouleimen, H. Lecocq. "A new efficient simulated annealing algorithm for the RCPSP and its multiple mode version", *EJOR* 149, pp. 268-281 (2003)

[30] M. Palpant, C. Artigues, P. Michelon. "LSSPER: solving RCPSP with large neighbourhood search", *Annals of O.R.*, 131, 1-4, pp. 237-257 (2004)

[31] R. Kolisch, S. Hartmann. "Experimental investigation of heuristics for the resource constrained scheduling problem: an update"; *EJOR* 174, pp. 23-37, (2006)

[32] R. V. Ahuja, T. L. Magnanti, J. B. Orlin. *Network Flows: Theory/Appl.*, Prentice, N.J (1993)

[33] M. Minoux. "Network synthesis and optimum design"; *Networks* 19, pp. 313-360, (1989)

[34] P. Tormos, A. Lova. "Project scheduling with time varying resource". *IJPR*, 38,16, pp. 3937-3952, (2000)