Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

Dynamic Online Planning and Scheduling Using a Static Invariant-Based Evaluation Model

Cédric Pralet and **Gérard Verfaillie**
ONERA – The French Aerospace Lab
F-31055, Toulouse, France
cedric.pralet,gerard.verfaillie@onera.fr

Abstract

Sequential decision-making under uncertainty often uses an approach in which a plan is built over a given horizon ahead using a deterministic model, the first decisions in this plan are applied, new information is acquired, the plan is adapted or rebuilt from scratch over a sliding horizon, and so on. This paper introduces a generic local search library that can be used in this context to quickly build and rebuild good quality plans. This library is built upon the notion of invariant used in constraint-based local search. Invariants allow temporal constraints, resource constraints, and criteria to be very quickly evaluated from a variable assignment and re-evaluated from a small change in this assignment. The paper also shows how the library can be used to reason on dynamic problem instances using a unique static problem model, without dynamic memory allocation. The approach is illustrated on a problem of data download under uncertainty about the volume of data, coming from the space domain.

1 Introduction

Planning and Scheduling (P&S) for autonomous systems often requires to take into account uncertainties about the system and environment state, unexpected events, and new request arrivals. Autonomous systems must also make decisions continuously (repeatedly) over a potentially long-term mission horizon.

The need to tackle such dynamic domains is already known in the planning community (Pollack and Horty 1999; desJardins et al. 1999), and several techniques were proposed for obtaining efficient and continuous replanning. These techniques include: (a) rationale-based monitors, which focus the replanning task on relevant changes in the environment (Veloso, Pollack, and Cox 1998); (b) HTN repair used in the CPEF framework (Myers 1999); (c) iterative plan repair used in CASPER to remove conflicts in plans (Chien et al. 2000); (d) dynamic maintenance and repair of plans used in IxTeT (Lemai and Ingrand 2004), based on POCL and STN reasoning; (e) techniques that take into account plan stability upon repair (Fox et al. 2006).

The need to handle dynamic domains is also already known in the scheduling community (Ovacikt and Uzsoy 1994; El Sakkout and Wallace 2000; Elkhyari, Guéret, and Jussien 2002), and more generally in combinatorial optimization (Hentenryck and Bent 2006; Verfaillie and Jussien 2005; Alba, Nakib, and Siarry 2012), with an emphasis on stability and robustness objectives and on incremental local search techniques.

This paper introduces a new generic framework for managing dynamic and continuous P&S problems. One of the main novelties of this framework is its ability to tackle *dynamic problems* using *static models*, which never change over the whole mission horizon. The use of static models is motivated by operational constraints regarding the software embedded on-board autonomous systems. More precisely, in the context of this study, the French space agency (CNES) asked us to define planning and replanning algorithms that do not use any dynamic memory allocation (no "new" instruction once the on-board software is initialized). This allows the exact memory size occupied by the planning process to be known beforehand. This also guarantees, by construction, the absence of memory leaks, memory heap overflows, or time loss due to online model creations.

Additionally, we must bear on reactivity needs and on strong limitations concerning computing resources available on board (below 100MHz CPU / 100MB RAM in typical space missions). Such limitations have an impact on P&S tools, as shown by the feedback on CASPER for mission EO-1 (Tran et al. 2004). Due to this operational context, the new framework we propose for dynamic P&S is based on local search, which is often able to quickly produce good-quality solutions using a low amount of memory.

The use of local search for planning is not new: see for instance iterative repair in ASPEN (Rabideau et al. 1999), enforced hill-climbing in FF (Hoffmann and Nebel 2001), stochastic local search in LPG (Gerevini, Saetti, and Serina 2003), or evolutionary algorithms in DAE (Bibaï et al. 2010). The main novelty here is that the local search approach proposed is based (1) on *dynamic Constraint Satisfaction Problems* (Mittal and Falkenhainer 1990), in which constraints can be active or not depending on the context, and (2) on *generic constraint-based local search* techniques, used in tools such as COMET and LocalSolver (Hentenryck and Michel 2005; Benoist et al. 2011). These tools are

built upon the notion of invariants, which allow combinatorial constraints, temporal constraints, resource constraints, and criteria to be very quickly evaluated from a variable assignment and re-evaluated from a small change in this assignment. We extend such invariant reasoning to integrate, in the context of a static model, P&S techniques defined for managing STN (Dechter, Meiri, and Pearl 1991) and time-dependent STN (Pralet and Verfaillie 2013). The new framework proposed is called InCELL for **In**variant-based **C**onstraint **E**va**L**uation **L**ibrary.

The paper aims at both presenting InCELL and InCELL at work on a concrete dynamic P&S problem. We first present this concrete problem, which is used all along the paper (Sect. 2). We then introduce the basis of InCELL (Sect. 3) and extensions for scheduling (Sect. 4). The presentation is focused on InCELL features that serve to model the case study. We then explain the way static models of InCELL are used for dynamic online P&S (Sect. 5). Finally, some experimental results are presented (Sect. 6).

2 Space Mission: a Data Download Problem

As an illustrative example, we use the problem of management of data downloads by satellites dedicated to the surveillance of electromagnetic sources at the Earth surface (Verfaillie et al. 2011). In this problem, surveillance plans (sequences of surveillance activities) are regularly built on the ground, uploaded to the satellites, and executed by them. However, the volume of data generated by any surveillance activity is not known in advance. Only a probability distribution on the actual volume is available. This distribution may have a large variance (typically, the actual volume may go from 1 to 1000). The actual volume of data generated by any surveillance activity a is only known at the end of a, when all the data generated by a is recorded in the satellite mass memory. Data downloads are then possible, but only within visibility windows of ground reception stations. Downloads can be performed concurrently with surveillance activities. The mission success depends on fair sharing among several entities that use the satellite, and on the so-called age of information (temporal distance between the realization of an acquisition and its delivery).

Due to the uncertainty on the volume of data, it becomes difficult to build data download plans completely off-line on the ground, as it is usual for many space missions. If maximum volumes of data are taken into account to build such plans, download windows may be under-used and mission return may be needlessly limited. If expected volumes are taken into account, some data may be lost due to actual volumes greater than expected. A solution consists in building data download plans on-line on board as actual volumes of data are progressively known. More precisely, a download plan is regularly built on board over a given set of visibility windows ahead, taking into account actual volumes for the activities that have been already performed and expected volumes for the others ; the first downloads in this plan are executed ; then, a new plan is built over a sliding horizon, at least when an activity ends and its actual volume is known, and so on. Another solution would consist in building a data download policy on the ground, using MDP (*Markov Decision Process* (Puterman 1994)), but the multi-dimensional continuous state space to be considered (volumes associated with each activity) together with the non-linear optimization criterion prevent us from using this approach.

3 InCELL: an Invariant-based Constraint EvaLuation Library

This section introduces the basis of InCELL, the new generic local search library which we aim at using for dynamic P&S.

3.1 Atomic and Non Atomic Variables

The first modeling entity of InCELL is the notion of *variable*. Variables are partitioned into *atomic variables*, whose value can be updated by the user, and *non atomic variables*, defined as functions of other variables. For P&S, atomic variables can model both controllable variables (also called *decision variables*), whose value is updated by the planner when searching for good-quality plans, and uncontrollable variables (also called *parameters*), whose value is updated when knowledge on the environment is available. InCELL handles only complete variable assignments at any step, as in constraint-based local search, that is every variable x in the model is always assigned, with value $x.val$.

We denote as $\mathbf{var}\{T\}$ the type of a variable x taking values of type T. Type T can be primitive ($bool$, int, $long$, $float$, $double$) or not. Constants of type $\mathbf{cst}\{T\}$ are special cases of variables of type $\mathbf{var}\{T\}$, whose value never changes. InCELL also offers more complex variable types, such as type $\mathbf{seq}\{T, N\}$ to represent sequences $[e_1, \ldots, e_n]$ of elements of type T whose size n is bounded by N.

3.2 Invariants

The concept of invariant is central in constraint-based local search. In the COMET system (Hentenryck and Michel 2005), it is defined as a one-way constraint $x \leftarrow exp$, where x is a variable and exp is a functional expression of other variables in the problem, such as $x \leftarrow sum(i \in [1..N])\, y_i$. Invariants are maintained automatically and incrementally during local moves. The incremental aspect reduces the amount of work required to recompute the outputs (left part) of invariants following small changes in the assignment of their inputs (right part). On the previous example, in case of change of y_k for some $k \in [1..N]$, it suffices to add to x the difference between the current value of y_k and its old value, instead of recomputing x from scratch. The only condition required for reevaluating invariants incrementally is that the set of invariants must be acyclic, so that a variable is not a function of itself.

The formal definition of invariants used in InCELL is given below. We make explicit the fact that an invariant can have several outputs (not necessarily a single variable on the left-hand side of the invariant). This will be useful to manage invariants provided later that use STN reasoning techniques.

Definition 1 *An* invariant *is a triple (I, O, f) with I and O sequences of variables called the* input *and* output *variables respectively, and f a function mapping assignments of I to assignments of O.*

A set of invariants Δ *is said to be* defined over a set of atomic variables V *iff all inputs of invariants in* Δ *belong to* V *or to outputs of invariants in* Δ*, and all outputs of invariants in* Δ *are pairwise distinct and do not belong to* V.

The directed graph G associated with a set of invariants Δ over V *is a graph containing nodes with input and output ports.* G *contains (a) one node per atomic variable* $v \in V$*; this node has no input port and a single output port labeled by* v*; (b) one node per invariant* $(I, O, f) \in \Delta$*; this node is labeled by* f *and has, for each* $i \in I$*, one input port labeled by* i*, and, for each* $o \in O$*, one output port labeled by* o*. In* G*, there is an arc from every output port labeled by a variable* x *to every input port also labeled by* x*. The set of invariants is said to be* acyclic *iff* G *is acyclic. In this case,* G *is called the* DAG of invariants.

Fig. 1 shows the DAG associated with a small set of invariants. In this figure, all invariants have a single output. InCELL offers a catalog of such single-output invariants: *boolean invariants* (output of type $\mathbf{var}\{bool\}$) built using logical operators ($\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$), comparators ($=$, $\neq$, $\leq$, $<$, $\geq$, $>$), and cardinality constraints; *numeric invariants* obtained via arithmetic operations (abs, opposite, min, max, ite, sum, weightedSum, minus, prod, div, ceil, floor, round, exp, sqrt, pow, sin...); invariants which are called *combinatorial invariants* in COMET, such as the *element* invariant.[1]

$$
\begin{array}{l}
V = \{\ b : \mathbf{var}\{bool\}, \\
\qquad x : \mathbf{var}\{double\}, \\
\qquad y : \mathbf{var}\{double\}, \\
\qquad 2 : \mathbf{cst}\{double\}\} \\
\mathbf{var}\{double\}\ z \leftarrow ite(b, x, y) \\
\mathbf{var}\{double\}\ t \leftarrow (x - y) \\
\mathbf{var}\{bool\}\ u \leftarrow (z < t) \\
\mathbf{var}\{double\}\ v \leftarrow (t + 2)
\end{array}
$$

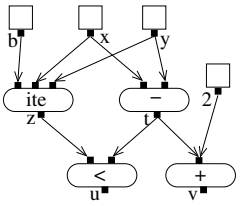

Figure 1: Set of invariants and associated DAG ($ite(b, x, y)$ stands for "if b then x else y")

Invariants are also available for handling sets of elements. For example, the invariant in Eq. 1 enforces that sequence s is a sorted version of the list of elements e_i in $elts$ that satisfy condition b_i. Elements are sorted according to v_i values in $vals$, using comparator $\preceq$. The invariant in Eq. 2 returns only the minimal elements. All invariants mentioned previously are handled in InCELL without dynamic memory allocation.

$$
\begin{array}{l}
elts = [e_1 : T, \ldots, e_N : T] \\
conds = [b_1 : \mathbf{var}\{bool\}, \ldots, b_N : \mathbf{var}\{bool\}] \\
vals = [v_1 : \mathbf{var}\{T'\}, \ldots, v_N : \mathbf{var}\{T'\}] \\
\mathbf{seq}\{T, N\}\ s \leftarrow sort_{\preceq}(elts, conds, vals) \qquad (1) \\
\mathbf{seq}\{T, N\}\ s' \leftarrow argmin_{\preceq}(elts, conds, vals) \qquad (2)
\end{array}
$$

Local Search Local moves can be performed by updating assignments of atomic variables. To handle invariants in this context, InCELL uses the same four basic methods as in LocalSolver (Benoist et al. 2011): (1) *init()*, for initializing the value of all outputs of invariants; (2) *eval()*, for reevaluating outputs in case of change of some inputs; (3) *commit()*, for accepting the state of the model as the new current state; (4) *rollback()*, for coming back to the state just after the last call to *commit()*.

In constraint-based local search, the main issue is to reevaluate all invariants with a minimum computational cost upon change in the assignment of atomic variables. To do this, the reevaluation process maintains a priority queue containing invariants to be reevaluated. This queue is ordered following a topological order of the DAG of invariants, which allows each invariant to be reevaluated at most once. The reevaluation queue initially contains invariants that are children of atomic variables whose assignment has changed. When an invariant is reevaluated, all children of modified outputs are added to the queue (if they were not already present in the queue). The reevaluation stops when the reevaluation queue is empty.

3.3 Example

We illustrate the previous elements on the data download problem. In the following, A denotes the maximum number of acquisitions that can be handled by the static model, W the maximum number of visibility windows, S the number of reception stations, E the number of entities that share the satellite, and P the number of priority levels associated with acquisitions. Fig. 2 gives a list of variables introduced to model the data download problem.

This list first contains parameters $hSta$ and $hEnd$, defining the sliding planning horizon. Constant features of the mission are then introduced: the emission rate of the download antenna ($dRate$), and the durations of data transfers between ground stations ($tDurs_{ij}$).

The next group of variables corresponds to features of reception station visibility windows $w \in [1..W]$, with their start time, their end time, and the index of the corresponding reception station (variables $wSta_w$, $wEnd_w$, and wSt_w).

Parameters are then defined for each acquisition $a \in [1..A]$: an entity requesting the acquisition (en_a), a priority degree (pr_a), a weight within this priority degree (w_a), an end time ($aEnd_a$), a delivery deadline after which a has no value ($dline_a$), a principal station to which a must be delivered (pSt_a), a volume occupied in mass memory (vl_a), and a download duration ($dDur_a$). The latter variable is not atomic: it is defined by the invariant of Eq. 3, as a function of the volume of a and of the download rate.

The download decision for a is represented by atomic variable $dWin_a$ whose value gives the index of the visibility window in which the download of a is planned (value 0 if no download). Several non atomic variables can be derived from $dWin_a$, including: (a) variable $dPres_a$, defined by the invariant of Eq. 4, which expresses that a is downloaded iff the download window chosen for a is not null; (b) variable dSt_a, defined by the invariant of Eq. 5, which expresses that the station to which a is downloaded is the station associated with window $dWin_a$; Eq. 5 implicitly uses an *element* invariant; (c) variable $tDur_a$, defined by the invariant of Eq. 6, which expresses that the duration required for transferring a on the ground to its principal station depends on transfer du-

[1]The *element* invariant takes as input a table of elements $tab : \mathbf{var}\{T\}[]$ and an index $idx : \mathbf{var}\{int\}$. It has a single output $x : \mathbf{var}\{T\}$ and enforces $x.val = tab[idx.val].val$.

Planning horizon		
$hSta$	$\mathbf{var}\{double\}$	start time of the horizon
$hEnd$	$\mathbf{var}\{double\}$	end time of the horizon
Mission features		
$dRate$	$\mathbf{cst}\{double\}$	download rate
$tDurs_{ij}$	$\mathbf{cst}\{double\}$	duration for data transfer from i to j, for stations $i, j \in [1..S]$
Reception station visibility windows $w \in [1..W]$		
$wSta_w$	$\mathbf{var}\{double\}$	start time of w
$wEnd_w$	$\mathbf{var}\{double\}$	end time of w
wSt_w	$\mathbf{var}\{int\}$	station associated with w
Acquisitions features $a \in [1..A]$		
en_a	$\mathbf{var}\{int\}$	entity requesting a
pr_a	$\mathbf{var}\{int\}$	priority level of a
w_a	$\mathbf{var}\{double\}$	weight of a
$aEnd_a$	$\mathbf{var}\{double\}$	end time of a
$dline_a$	$\mathbf{var}\{double\}$	deadline for delivering a
pSt_a	$\mathbf{var}\{int\}$	principal station for a
vl_a	$\mathbf{var}\{double\}$	volume of a
$dDur_a$	$\mathbf{var}\{double\}$	duration of the download of a
Download decisions for $a \in [1..A]$		
$dWin_a$	$\mathbf{var}\{int\}$	index of the window chosen for downloading a
$dPres_a$	$\mathbf{var}\{bool\}$	presence of the download of a
dSt_a	$\mathbf{var}\{int\}$	index of the ground station to which a is downloaded
$tDur_a$	$\mathbf{var}\{double\}$	transfer duration of a on ground
$dSta_a$	$\mathbf{var}\{double\}$	start time of the download of a
$dEnd_a$	$\mathbf{var}\{double\}$	end time of the download of a
Activations		
$wAct_w$	$\mathbf{var}\{bool\}$	activation status of $w \in [1..W]$
$aAct_a$	$\mathbf{var}\{bool\}$	activation status of $a \in [1..A]$
Planning algorithm		
$cand_a$	$\mathbf{var}\{bool\}$	candidate status of $a \in [1..A]$ for insertion in the plan
$bestA$	$\mathbf{seq}\{int, A\}$	acquisitions of highest priority that are active and candidate for insertion

Figure 2: Variables for the data download problem

rations $tDurs_{ij}$ and on the station to which a is downloaded. Variables $dSta_a$ and $dEnd_a$ in Fig. 2 give the precise start and end times of the download of a. They will be defined by more complex invariants given later in the paper.

The next two sets of variables given in Fig. 2, $wAct_w$ for every window $w \in [1..W]$ and $aAct_a$ for every acquisition $a \in [1..A]$, will allow us to make acquisitions and visibility windows active or not in the model depending on whether they are involved in the current planning horizon. Said differently, they will allow us to reason in a *dynamic CSP* fashion. Activations/inactivations will be useful in the quest for a static model to handle dynamic problems. Let us also point out that all features describing download windows and acquisitions are defined as variables and not as constants. This will allow us to recycle these elements each time the planning horizon slides (more details in Sect. 5).

Last, for the planning algorithm, special variables $cand_a$ are introduced to indicate whether acquisition a is candidate for insertion in the plan (when the insertion of a fails, it is marked as non-candidate). The set of acquisitions that are active, candidate for insertion, and whose priority is maximal ($bestA$) can then be automatically maintained via the invariant of Eq. 7 (an *argmin* invariant, as in Eq. 2).

$$\forall a \in [1..A],\ dDur_a \leftarrow vl_a / dRate \tag{3}$$

$$dPres_a \leftarrow (dWin_a \neq 0) \tag{4}$$

$$dSt_a \leftarrow wSt_{dWin_a} \tag{5}$$

$$tDur_a \leftarrow tDurs_{dSt_a, pSt_a} \tag{6}$$

$$bestA \leftarrow argmin_{\leq}([1, \ldots, A], [aAct_1 \wedge cand_1, \ldots, aAct_A \wedge cand_A], [pr_1, \ldots, pr_A]) \tag{7}$$

3.4 Constraints

Another modeling entity used in constraint-based local search is the notion of constraint. For InCELL, a constraint is a particular invariant whose evaluation stops the evaluation process when the constraint is violated. An invariant enforcing a constraint can have outputs as well as no output at all. For example, the constraint in Eq. 8 returns an inconsistency upon evaluation iff variable b takes value false.

$$constraint(b : \mathbf{var}\{bool\}) \tag{8}$$

Eq. 9 and 10 are examples of two basic constraints to be satisfied for the data download problem. The former imposes that the window chosen for downloading an acquisition a must be active. The latter imposes a domain constraint on the variable giving the station used for downloading a.

$$\forall a \in [1..A],\ constraint(dPres_a \rightarrow wAct_{dWin_a}) \tag{9}$$

$$constraint(dSt_a \in [0..S]) \tag{10}$$

3.5 Criterion / Criteria

Complex criteria can be modeled in InCELL thanks to invariants. Minimizing makespan or tardiness is just a possible option. For the data download mission, the criterion to be optimized is expressed using the additional invariants given in Fig. 3. The goal here is not to give a precise description of this criterion, which tries to combine download efficiency and fairness between entities, but just to give the intuition. The criterion first defines age_a as the temporal distance between the end of acquisition a and the delivery date of a to its principal station. From this age, we compute a corrected weight wc_a, by multiplying the weight of a by a factor equal to 1 if a is delivered immediately after its realization, and decreased by a factor of 2 every K (a constant) time units.

From corrected weights, it is possible to define $u_{p,e}$ the utility obtained at priority level p for an entity e. This utility takes into account a past utility $uPast_{p,e}$ obtained in the history of the satellite. In order to combine utilities over all entities and obtain an aggregated utility level u_p at each priority level p, we use formulas involving a parameter $par \neq 0$ that balances the trade-off between utilitarism (case $par = 1$) and equity between entities (case $par = -\infty$). At priority level 3, the formula additionally uses a constant quota Q_e associated with each entity e. As any improvement at priority level p is preferred to any improvement at level $p' > p$, the objective is then to maximize vector of utilities (u_1, u_2, u_3) using a lexicographic order.

$\forall a \in [1..A]$
$age_a \leftarrow ite(dPres_a, dEnd_a + tDur_a - aEnd_a, 0)$
$wc_a \leftarrow w_a \cdot ite(dPres_a, 2^{-age_a/K}, 0)$
$\forall p \in [1..P], \forall e \in [1..E]$,
$u_{p,e} \leftarrow ite(bPast_{p,e} \vee (\vee_{a \in [1..A] \mid pr_a = p, en_a = e} aAct_a),$
$uPast_{p,e} + \sum_{a \in [1..A] \mid pr_a = p, en_a = e} wc_a,$
$1)$
$u_1 \leftarrow (1/E \cdot \sum_{e \in [1..E]} u_{1,e}^{par})^{1/par}$
$u_2 \leftarrow (1/E \cdot \sum_{e \in [1..E]} u_{2,e}^{par})^{1/par}$
$u_3 \leftarrow (\sum_{e \in [1..E]} Q_e \cdot (u_{3,e}/Q_e)^{par})^{1/par}$

Figure 3: Mission criterion for three priority levels ($P = 3$)

4 Scheduling Abstractions

Similarly to COMET and to constraint-based schedulers like ILOG CP Optimizer, InCELL contains higher level P&S abstractions useful for making the modeling task easier.

4.1 Time-points

Type $\mathbf{Time}$ is used to represent *time-points*. A time-point $t : \mathbf{Time}$ is defined by two basic variables of type $\mathbf{var}\{double\}$ that represent the earliest and the latest realization dates associated with t.

For the data download problem, we can introduce, for each acquisition $a \in [1..A]$, two time-points:

- $ts_a : \mathbf{Time}$: floating start time of download for a;
- $te_a : \mathbf{Time}$: floating end time of download for a.

4.2 Intervals

Constant Intervals Type $\mathbf{ItvConst}$ is used to represent temporal intervals that have no flexibility on their realization dates. An interval $itv : \mathbf{ItvConst}$ is defined by three basic variables: a presence variable $pres(itv) : \mathbf{var}\{bool\}$, a start time variable $sta(itv) : \mathbf{var}\{double\}$, and an end time variable $end(itv) : \mathbf{var}\{double\}$. The presence variable is useful for scheduling problems in which some activities are optional.

In the data download problem, the constant interval associated with each station visibility window $w \in [1..W]$ can be defined as $wItv_w = (wAct_w, wSta_w, wEnd_w)$. The list of active station visibility windows ordered by increasing start times can be obtained using the invariant of Eq. 11.

$$\mathbf{seq}\{\mathbf{ItvConst}, W\}\ wItvSort \leftarrow sort_{\leq}(\ [wItv_1, \ldots, wItv_W], [wAct_1, \ldots, wAct_W], [wSta_1, \ldots, wSta_W]) \quad (11)$$

Floating Intervals Type $\mathbf{Itv}$ is used to represent intervals that have flexibility on their start/end times. An interval $itv : \mathbf{Itv}$ is defined by three elements: a presence variable $pres(itv) : \mathbf{var}\{bool\}$, a start time-point $sta(itv) : \mathbf{Time}$, and an end time-point $end(itv) : \mathbf{Time}$.

The floating interval associated with the download of acquisition $a \in [1..A]$ can be defined as $dItv_a = (dPres_a, ts_a, te_a)$. For the data download problem, it is also possible to define a sequence of floating intervals:

- $\mathbf{seq}\{\mathbf{Itv}, A\}$ *dSeq*: sequence of download intervals (size bounded by A, the maximum number of acquisitions).

4.3 Unary Temporal Constraints

Type $\mathbf{After}$ (resp. $\mathbf{Before}$) is used to represent temporal constraints over the earliest (resp. latest) realization date of a time-point. Basically, a constraint c of type $\mathbf{After}$ (resp. $\mathbf{Before}$) is defined by an invariant *after(p,x,d)* (resp. *before(p,x,d)*) that has three inputs:

- a boolean presence variable $p : \mathbf{var}\{bool\}$,
- a reference to a time-point $x : \mathbf{var}\{\mathbf{Time}\}$,
- a double variable $d : \mathbf{var}\{double\}$.

It represents conditional constraint $[p.val]\, x.val \geq d.val$ (resp. $[p.val]\, x.val \leq d.val$), which is satisfied iff $x.val \geq d.val$ (resp. $x.val \leq d.val$) holds whenever $p.val$ is true. For readability reasons, $after(p, x, d)$ (resp. $before(p, x, d)$) is also directly written as $[p]\, x \geq d$ (resp. $[p]\, x \leq d$).

For the data download problem, such invariants are used to define temporal constraints $c_{1,a}$ and $c_{2,a}$ in Fig. 4. Constraint $c_{1,a}$ imposes that an acquisition cannot be downloaded before the end of its realization, before the start of its associated station visibility window, or before the start of the planning horizon. The right part of $c_{1,a}$ is actually defined by a hierarchy of invariants which are automatically reevaluated when required, for instance in case of change of $dWin_a$. Constraint $c_{2,a}$ imposes that an acquisition must be delivered before its deadline, and that its download must end before the end of its associated station visibility window and before the end of the planning horizon.

$\forall a \in [1..A]$,
$\mathbf{After}\, c_{1,a} \leftarrow [dPres_a]\, ts_a \geq \max(aEnd_a, wSta_{dWin_a}, hSta)$
$\mathbf{Before}\, c_{2,a} \leftarrow [dPres_a]\, te_a \leq \min(dline_a - tDur_a, wEnd_{dWin_a}, hEnd)$
$\mathbf{Dist}\, c_{3,a} \leftarrow [dPres_a]\, te_a - ts_a \leq dDur_a$
$\mathbf{Dist}\, c_{4,a} \leftarrow [dPres_a]\, ts_a - te_a \leq -dDur_a$
$\mathbf{Dist}[]\, c_5 \leftarrow orderedNoOverlap(dSeq)$

Figure 4: Temporal constraints on data downloads

4.4 Temporal Distance Constraints

Type $\mathbf{Dist}$ is used to represent distance constraints. A distance constraint is defined by an invariant *dist(p,x,y,d)* that has four inputs:

- a boolean presence variable $p : \mathbf{var}\{bool\}$,
- two references to time-points $x, y : \mathbf{var}\{\mathbf{Time}\}$,
- a double variable $d : \mathbf{var}\{double\}$ (distance bound).

It represents conditional constraint $[p.val]\, y.val - x.val \leq d.val$, which can be used to model precedence constraints, precedence constraints with setup times, or duration constraints. For readability reasons, $dist(p, x, y, d)$ is also written as $[p]\, y - x \leq d$. Note that a distance constraint $[pres(itv)]\, sta(itv) - end(itv) \leq 0$ is implicitly associated with each floating interval itv. Also, $\mathbf{After/Before}$ constraints could be handled as distance constraints with regard to a reference temporal position. They are treated separately for computational efficiency.

For the data download problem, invariant *dist* is used to define temporal constraints $c_{3,a}$ and $c_{4,a}$ in Fig. 4. These constraints express that the temporal distance between the start and the end of the download of a must be equal to duration $dDur_a$, which is itself defined by an invariant ($dDur_a \leftarrow vl_a / dRate$).

Distance constraints can also be obtained using the multiple-output invariant of Eq. 12. Given a sequence of floating intervals $s = [itv_1, \ldots, itv_n]$ of size bounded by N, this invariant returns as an output a table of $N-1$ distance constraints. The invariant guarantees that the conjunction of these $N-1$ constraints is always equivalent to $\wedge_{i \in [1..n-1]}(end(itv_i) \leq sta(itv_{i+1}))$, *i.e.* to the fact that intervals in the sequence are temporally ordered and do not overlap.

$$\mathbf{Dist}[\,]\ c \leftarrow orderedNoOverlap(s : \mathbf{seq}\{\mathbf{Itv}, N\}) \quad (12)$$

For the data download problem, the invariant of Eq. 12 is used in Fig. 4 for constraint c_5, to express that data downloads in download sequence $dSeq$ must not overlap.

From a technical point of view, in order to obtain a static model, the $orderedNoOverlap$ invariant is handled without dynamic memory allocation, even if the set of real temporal constraints it induces is dynamic. More precisely, the invariant maintains a set of $N-1$ distance constraints $c_i : [p_i]\, y_i - x_i \leq 0$ with, for every $i \in [1..N-1]$, $p_i : \mathbf{var}\{bool\}$ a boolean variable and $x_i, y_i : \mathbf{var}\{\mathbf{Time}\}$ two references to time-points. For each interval itv that has a successor in the sequence, we maintain an index $k(itv)$ such that $c_{k(itv)}$ corresponds to the temporal constraint between itv and its successor in the sequence. If an interval itv'' is inserted between itv and itv': (1) we update constraint $c_{k(itv)} : [p]\, y - x \leq 0$ by setting $x.val = sta(itv'')$, and (2) we get an unused constraint c_j (such that $p_j.val$ equals *false*) and we update c_j by setting $p_j.val = true$, $x_j.val = sta(itv')$, $y_j.val = end(itv'')$. Similar operations are used when an interval is inserted at the beginning or at the end of the sequence, or when an interval is removed from the sequence.

4.5 Simple Temporal Network (STN) Invariant

Consistent realization dates for time-points can be obtained using a so-called *STN invariant* $stnAssign_{Tp}$, as defined in Eq. 13. This invariant is parametrized by a table Tp containing time-points. It takes as input sets of temporal constraints of type $\mathbf{After}$, $\mathbf{Before}$, and $\mathbf{Dist}$. It returns as an output a table et of $\mathbf{var}\{double\}$ elements. When it is evaluated, the invariant checks consistency and ensures that the value of et_i is equal to the earliest time associated with time-point Tp_i in a consistent schedule. These operations are performed in one shot for all changes in parent temporal constraints.

$$\mathbf{var}\{double\}[\,]\ et \leftarrow stnAssign_{Tp}(\quad (13)$$
$$aCtrs : \mathbf{After}[\,], bCtrs : \mathbf{Before}[\,], dCtrs : \mathbf{Dist}[\,])$$

To model the data download problem, we use an STN invariant which takes as input all constraints defined in Fig. 4. The table of time-points considered is $Tp = [ts_1, te_1, \ldots, ts_A, te_A]$ (table of floating start and end times of downloads). Table et returned by the invariant is $[dSta_1, dEnd_1, \ldots, dSta_A, dEnd_A]$. This means that elements $dSta_a$, $dEnd_a$ given in Fig. 2 are actually defined by an invariant: they are not atomic variables whose value can be fixed freely, since they must satisfy temporal constraints. The values of these variables will be automatically computed using STN reasoning techniques. In particular, given an acquisition a, if the volume vl_a of a is changed, the download duration of a will be updated via the invariant defining $dDur_a$, temporal constraints $c_{3,a}$, $c_{4,a}$ in Fig. 4 will be reevaluated, and the earliest realization dates for downloads will be recomputed via invariant $stnAssign$.

The latter is maintained incrementally using standard STN techniques (Cervoni, Cesta, and Oddi 1994; Cesta and Oddi 1996; Gerevini, Perini, and Ricci 1996; Shu, Effinger, and Williams 2005). These techniques allow us to recompute feasible bounds of time-points upon strengthening, weakening, deletion, or removal of constraints. More precisely, the approach uses constraint propagation, maintains propagation chains in order to detect negative cycles and to depropagate constraints incrementally, and it uses a decomposition of the distance graph of the STN into strongly connected components in order to determine a good constraint propagation order (Pralet and Verfaillie 2013).

Even if the set of constraints which hold over a time-point x may be dynamic during search (*e.g.* due to the $orderedNoOverlap$ invariant), the $stnAssign$ invariant is implemented without dynamic memory allocation. Technically speaking, we use linked list structures that allow dynamic STN to be handled using a space complexity linear in the number of time-points and constraints. All internal data structures used by the STN invariant are defined as variables, over which *commit()* and *rollback()* operations are used in case of acceptance or rejection of a local move.

A part of the DAG of invariants obtained for the data download problem is shown in Fig. 5. In this DAG, it is possible to see that the STN invariant is an invariant as all other quantities, and that it has both ancestors and descendants.

4.6 Discussion

Other planning and scheduling abstractions are available in InCELL, including cumulative resources, energy resources with production/consumption rates, and a richer class of temporal constraints, such as those used in time-dependent scheduling (Cheng, Ding, and Lin 2004; Gawiejnowicz 2008; Pralet and Verfaillie 2013), in which the duration of an activity may depend on its start time.

With regard to existing works, COMET offers additional elements for fast prototyping of local search algorithms. It also has the notion of differential invariants, which is useful to determine how much a constraint is violated. On the other side, the core of the temporal model of COMET is a precedence graph, whereas InCELL handles the more general class of STN constraints. InCELL also directly and explicitly includes STN reasoning within the DAG of invariants. This allowed us to define activities whose duration is a variable, temporal constraints whose features (presences, time points, and bounds) are given by invariants, and to have a direct access to earliest download end times, used in the

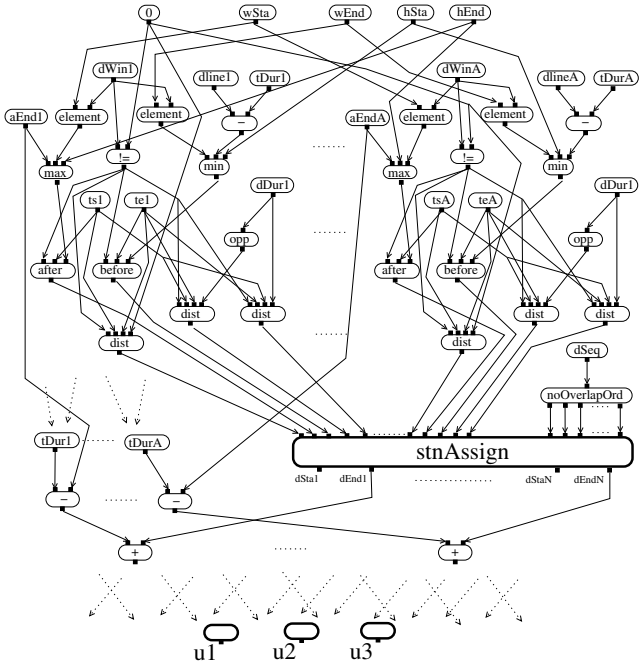

Figure 5: Overview of the DAG of invariants; in bold: STN invariant, and outputs giving vector of utilities (u_1, u_2, u_3)

criterion. Last, InCELL puts the emphasis on the management of memory, in order to obtain models whose memory size remains static during search. As far as we know, such a guarantee is not offered by COMET.

5 Continuous Autonomous Planning

We now show InCELL at work on the data download problem. This problem is *dynamic* for two reasons:

1. the actual volume of acquisitions is known at run-time, when acquisitions end; if this actual volume is greater than the estimated volume considered at planning time, the current plan may become temporally inconsistent; otherwise, there may be some place for plan re-optimization;
2. the satellite must consider new acquisitions and new station visibility windows when the planning horizon is slided forward, or when new acquisition plans are received from control ground stations.

The satellite must continuously plan downloads over the *mission horizon*, which can range from a few weeks to several months or years of continuous and autonomous planning and scheduling. This mission horizon must not be mistaken for the *planning horizon* used to define the time window over which the planner reasons to produce decisions.

5.1 Static Models for Dynamic Problems

In the following, we show how it is possible to plan continuously using the invariant-based model defined in the previous sections, without creating any variable, any constraint, or any object over the mission horizon.

The basic idea is to reason over a sliding planning horizon and to *recycle* objects that become unused over this horizon. For acquisitions, recycling means reusing old acquisitions, that are not in the catalog of acquisitions to be downloaded anymore. The attributes of such acquisitions (priority, entity, volume...) and the temporal constraints associated with them can be reused and updated freely. Doing so, an acquisition $i \in [1..A]$ can at some step be associated with entity B and have priority 2, and later on be recycled and associated with entity C and have priority 1. Similarly, a visibility window $w \in [1..W]$ can at some step be associated with station 1 in interval $[253, 814]$, and later on recycled and associated with station 2 in interval $[2460, 2975]$.

An important point is that objects of the model are not renumbered when other objects become unused, *e.g.* window number 5 may be used while window number 1 is not. In this case, window number 1 is a candidate for recycling. Avoiding renumbering simplifies invariants reevaluations, since for instance changing the index of a visibility window would also change $dWin_a$ for every acquisition a that uses this visibility window, and hence change temporal constraints and force useless recomputations at the level of the STN invariant.

To make the recycling of objects easier, InCELL contains special variables, of type $\mathbf{pool}\{T\}$, for managing *pools of objects* of type T. For the data download problem, we use one pool containing all acquisition objects, and one pool containing all station visibility windows objects. For pools of type T, function $addInstance(e:T)$ is used at initialization to populate the pool, function $newInstance()$ is used to return an unused object of type T, and function $freeInstance(e:T)$ is used to put back an unused object in the pool. Elements of type $\mathbf{pool}\{T\}$ can also watch boolean usage variables such as $aAct_a$ and $wAct_w$ (see Fig. 2), in order to automatically maintain the set of unused objects. Pools of objects can therefore be seen as static memory allocation managers. Commit and rollback operations can be performed over pools, as over any other InCELL element.

5.2 Online Planning and Replanning

To reason over the invariant-based constraint model, we use a Local Search strategy (LS). InCELL is used in LS both for evaluating the impact of download decisions and for modifying problem data when new information is available. LS is encapsulated within a deliberative layer which produces plans and sends them to a reactive layer. The latter is responsible for sending decisions to the executive. It also contains a simple decision rule to be used when LS does not manage to produce a first plan before the decision deadline, or when the size of the dynamic problem becomes temporarily too large for the static model used within LS.

LS builds and repairs plans over planning horizon $[hSta, hEnd]$. It uses Non Chronological Greedy Search (NCGS). The latter iteratively inserts download activities in the current plan, until no new insertion is possible. Each iteration corresponds to the selection of one candidate acquisition a of highest priority, one visibility window w, and one position k in the download sequence. Triple (a, w, k) is chosen so as to maximize ratio $\Delta u / dDur_a$ between (1) the vari-

ation of utility Δu obtained by inserting a in window w and at position k in the download sequence, and (2) the duration of the download of a (standard knapsack insertion heuristics "value of an object divided by its size"). If no consistent values w, k are found, a becomes non-candidate.

Replanning/repair occurs (1) when a new acquisition plan is received from control ground stations, (2) when the horizon slides and covers new download windows, and (3) when an acquisition ends. In the first two cases, NCGS is used to refill the current plan. In the third case, when an acquisition ends and its volume changes, InCELL incrementally recomputes the consistency of the current plan. In case of consistency, if the real volume is lower than the volume considered for planning, NCGS is applied again to try and insert new downloads. In case of inconsistency, a consistent plan is recovered by removing the download of a, and NCGS is applied on the obtained plan. Better repair strategies could be designed, e.g. by analyzing the explanations for temporal inconsistency. Once the current plan is repaired/reoptimized, LS sends it to the reactive layer. A deeper replanning is then performed by reapplying NCGS from an empty plan. Restarts from such empty plans and randomization in NCGS can also be used to diversify search.

6 Experiments

Experiments were performed on a one week scenario provided by CNES. This scenario involves $S = 2$ ground stations, $E = 2$ entities, and $P = 3$ priority degrees. Over this horizon, the acquisition plan contains 1484 acquisitions and 96 station visibility windows. The length of the planning horizon is set to 30 minutes. The static model created can simultaneously handle a maximum number of $A = 400$ acquisitions and $W = 7$ station visibility windows over the horizon. Even if $A < 1484$ and $W < 96$, the static model manages to handle the scenario considered thanks to object recycling. The number of variables and invariants used by the model with these settings is 16884, and the whole planner occupies around 18MB of RAM. In the criterion, parameter par is set to value -3, which corresponds to a trade-off between utilitarism and equity between entities.

Local search strategy LS based on InCELL is compared with on-board approach PB1 defined in (Verfaillie et al. 2011). PB1 uses the same strategy as NCGS (non chronological greedy search), with the only difference that when PB1 inserts a new download in the plan, the start and end times of this download become *fixed* (no flexibility). This may induce holes between downloads and under-use of downlink windows. Also, PB1 is not generic (it is optimized for the mission) and it systematically replans from scratch.

On an Intel i5-520 1.2GHz, 4GBRAM, the mean CPU time consumed by LS for each plan repair is 6ms, and the mean CPU time to plan from scratch is 50ms. For PB1, the mean CPU time to replan from scratch is about 1ms.

For the experiments, we forced LS to systematically replan from scratch, as PB1. Fig. 6 gives the results obtained. For both methods, all downloadable acquisitions of priority 1 are downloaded. The proportion of acquisitions downloaded decreases with the priority degree, which is not surprising due to the lexicographic ordering on vectors of utilities (u_1, u_2, u_3). The comparison of $u_{p,e}$ elements also shows that plans produced are quite fair between entities, except for priority level 3 for which entity quotas interfere. Compared to PB1, LS improves on the number of downloads, especially for priority 3, and better utilizes download windows (see tables in the upper and middle parts). This essentially comes from the STN invariant, which allows temporal flexibility to be maintained. Last, as shown by the graphs at the bottom of Fig. 6, LS improves on the age of acquisitions for all priority levels.

	downloads	% window use	u_1	u_2	u_3
PB1	1121/1484	95.77%	0.925	0.857	0.608
LS	1188/1484	96.38%	0.929	0.852	0.653

priority	entity	#a (#i)	PB1		LS	
			#d	$u_{p,e}$	#d	$u_{p,e}$
1	A	98 (5)	93	0.930	93	0.933
	B	28 (1)	27	0.940	27	0.943
	C	14 (1)	13	0.908	13	0.911
2	A	294 (16)	247	0.814	248	0.821
	B	84 (3)	73	0.846	71	0.824
	C	42 (2)	40	0.925	40	0.928
3	A	645 (39)	439	0.608	495	0.686
	B	183 (10)	126	0.611	112	0.539
	C	96 (0)	63	0.607	89	0.867

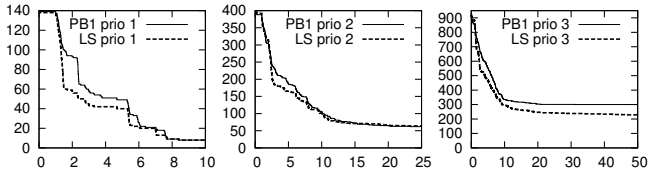

Figure 6: Comparison between LS and PB1. Upper part: global results; middle part: detailed results per priority level and per entity (#a = nb of acquisitions, #i = nb of acquisitions which end after the end of the last station visibility window of the scenario, #d = nb of downloads); bottom part: repartition of the age of acquisitions (for each priority level, the y-axis represents the number of acquisitions whose age is greater than age limit in hours given on the x-axis)

7 Conclusion and Perspectives

This paper presented a new framework for dynamic P&S. The contributions are: (1) an integration of existing techniques (constraint-based local search, dynamic constraint satisfaction, and STN reasoning) for the sake of continuous dynamic P&S using a static model; (2) a reduction of the gap between generic planning/replanning libraries and critical embedded software subject to implementation constraints concerning static data structures; (3) the treatment of a real-world application, using some knowledge-engineering for modeling activities, constraints, and non-standard optimization criteria. In the future, we plan to integrate some generic meta-heuristics for local search and to interleave more finely InCELL with the executive layer.

References

Alba, E.; Nakib, A.; and Siarry, P., eds. 2012. *Metaheuristics for dynamic optimization*. Springer.

Benoist, T.; Estellon, B.; Gardi, F.; Megel, R.; and Nouioua, K. 2011. Localsolver 1.x: a black-box local-search solver for 0-1 programming. *4OR: A Quarterly Journal of Operations Research* 9(3):299–316.

Bibaï, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *Proc. of ICAPS'10*, 18–25.

Cervoni, R.; Cesta, A.; and Oddi, A. 1994. Managing dynamic temporal constraint networks. In *Proc. of AIPS'94*, 13–18.

Cesta, A., and Oddi, A. 1996. Gaining efficiency and flexibility in the simple temporal problem. In *Proc. of TIME'96*, 45–50.

Cheng, T.; Ding, Q.; and Lin, B. 2004. A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research* 152:1–13.

Chien, S.; Knight, R.; Stechert, A.; R.Sherwood; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proc. of AIPS'00*, 300–307.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

desJardins, M.; Durfee, E.; Ortiz, C.; and Wolverton, M. 1999. A survey of research in distributed continual planning. *AI Magazine* 20(4):13–22.

El Sakkout, H., and Wallace, M. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints* 5(4):359–388.

Elkhyari, A.; Guéret, C.; and Jussien, N. 2002. Conflict-based repair techniques for solving dynamic scheduling problems. In *Proc. of CP'02*, 702–707.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: replanning versus plan repair. In *Proc. of ICAPS'06*, 212–221.

Gawiejnowicz, S. 2008. *Time-dependent scheduling*. Springer.

Gerevini, A.; Perini, A.; and Ricci, F. 1996. Incremental algorithms for managing temporal constraints. In *Proc. of ICTAI'96*, 360–365.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research* 20(1):239–290.

Hentenryck, P. V., and Bent, R. 2006. *Online stochastic combinatorial optimization*. MIT Press.

Hentenryck, P. V., and Michel, L. 2005. *Constraint-based local search*. MIT Press.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *Proc. of AAAI'04*, 617–622.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proc. of AAAI'90*, 25–32.

Myers, K. 1999. CPEF: A continuous planning and execution framework. *AI Magazine* 20(4):63–69.

Ovacikt, I., and Uzsoy, R. 1994. Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research* 32(6):1243–1263.

Pollack, M., and Horty, J. 1999. There's more to life than making plans: plan management in dynamic multiagent environments. *AI Magazine* 20(4):71–83.

Pralet, C., and Verfaillie, G. 2013. Time-dependent simple temporal networks: properties and algorithms. *RAIRO Operations Research*.

Puterman, M. 1994. *Markov decision processes, discrete stochastic dynamic programming*. John Wiley & Sons.

Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative repair planning for spacecraft operations using the ASPEN system. In *Proc. of i-SAIRAS'99*, 99–106.

Shu, I.; Effinger, R.; and Williams, B. 2005. Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. In *Proc. of ICAPS'05*, 252–261.

Tran, D.; Chien, S.; Rabideau, G.; and Cichy, B. 2004. Flight software issues in onboard automated planning: lessons learned on EO-1. In *Proc. of IWPSS-04*.

Veloso, M.; Pollack, M.; and Cox, M. 1998. Rationale-based monitoring for planning in dynamic environments. In *Proc. of AIPS'98*, 171–179.

Verfaillie, G., and Jussien, N. 2005. Constraint solving in uncertain and dynamic environments. *Constraints* 10(3):253–281.

Verfaillie, G.; Infantes, G.; Lemaître, M.; Théret, N.; and Natolot, T. 2011. On-board decision-making on data downloads. In *Proc. of IWPSS-11*.