Improving Robustness of Spacecraft Downlink Schedules

Angelo Oddi *Member, IEEE,* and Nicola Policella

Abstract

In the realm of scheduling problems different sources of uncertainty can invalidate the planned solutions: unpredictability of activity behaviors, machine breakdowns, new activities to be served, and so on. In this paper we are concerned with the generation of high quality downlink schedules in a spacecraft domain in presence of a high degree of uncertainty. In particular, we refer to a combinatorial optimization problem called MARS-EXPRESS Memory Dumping Problem (MEX-MDP), which arose in the European Space Agency program MARS-EXPRESS. A MEX-MDP consists in the generation of dumping commands to maximize the downloads of data sets from the satellite to the ground. The domain is characterized with several kinds of constraints - such as, bounded on-board memory capacities, limited communication windows over the downlink channels, deadlines and ready times imposed by the payload requirements - and different sources of uncertainty - e.g., the amount of data generated at each scientific observation or the channel data rate.

In this paper we tackle this problem by using a reduction of the MEX-MDP to a Max-Flow problem: the former problem has a solution when the maximum flow in the latter equates the total amount of data to dump. Given this reduction, we introduce a novel definition of solution robustness based on the utilization of the on-board memory, as well as an iterative procedure to improve solution quality. The key idea behind this approach is that the lower the *peaks* in memory utilization, the higher the ability cope with unexpectedly large amount of data.

Index Terms

scheduling, uncertainty, robustness, max-flow.

I. INTRODUCTION

In a space domain, as well as many other applicative domains, the usefulness of a schedule is limited by its brittleness. Though a schedule offers the potentials for optimized operations, it must in fact be executed exactly as planned to achieve these potentials. In practice, this is generally made difficult in dynamic execution environments, where unexpected events quickly invalidate the schedule's predictive assumptions and the validity of the schedule's prescribed actions is continuously brought into question. The lifetime of a schedule tends to be very short, and hence its optimizing advantages are generally not realized. In this paper, we refer to a combinatorial optimization problem which arose in the European Space Agency program MARS-EXPRESS: the MARS-EXPRESS Memory Dumping

Angelo Oddi and Nicola Policella are with the Planning & Scheduling Team, (http://pst.istc.cnr.it). Institute for Cognitive Science and Technology, Italian National Research Council (ISTC-CNR), I-00137 Rome, Italy (e-mail: {firstname.lastname}@istc.cnr.it).

Problem (MEX-MDP). The MEX-MDP problem involves the process of automating the memory dump operations of both science and housekeeping data. Problems similar to MEX-MDP can arise in satellite domains such as the ones described in [1], [2]. These works concern a set of Earth observation operations to be allocated in time under a set of mandatory constraints such as: avoiding overlapping images, allowing sufficient transition times (or *setup* times), and allowing bounded instantaneous data flow and on-board limited recording capacity.

In this paper we model the MEX-MDP problem through a Max-Flow paradigm. Moreover, after recognizing peaks of memory utilization as sources of brittleness in the final schedule, we increase the solution's robustness proposing, thanks to the Max-Flow reduction, an iterative *leveling* algorithm that explores different distributions of the dumping operations by *flattening* the peaks in memory utilization.

The paper is organized as follows. We start describing the MARS-EXPRESS domain, next we provide a formal introduction of the MEX-MDP problem and a definition of solution's robustness for this problem. Therefore, given a MEX-MDP instance, a novel model based on the flow network paradigm is described. Thanks to this model, the problem can be solved as a max-flow instance. Based on this result we present an iterative max-flow solver that aims at improving the schedule's robustness. Finally, the paper ends presenting an experimental evaluation and a discussion of future developments that this method entails.

II. CONTEXT OF THE WORK

MARS-EXPRESS is one of the first missions of the ESA long-term Scientific Programme Horizons 2000 launched on last June 2003. As it is well known, the space probe is currently orbiting around the Red Planet and operating by using different scientific payloads. The MARS-EXPRESS domain is characterized by several kinds of constraints - such as bounded on-board memory capacities, limited communication windows over the downlink channels, deadlines and ready times imposed by the payload requirements - and different sources of uncertainty - e.g., the amount of data generated at each scientific observation or the channel data rate. These constraints make MARS-EXPRESS program a challenging and interesting domain for research in automated problem solving [3], [4], [5].

In this work we refer to an abstract *core* model of the MARS-EXPRESS Memory Dumping problem (MEX-MDP), which allows us to introduce the Max-Flow model of the problem and to recognize one of the main source of brittleness for a downlink schedule, i.e. peaks of data volumes stored in the on-board memory. In fact, generating high quality schedules for spacecraft downlink scheduling problems can hardly be seen as a single objective optimization problem, but rather as an optimization problem involving multiple, conflicting and non-commensurate criteria.

In the MEXAR project [3] we have started a research path in this direction with the main goal of defining a Decision Support System (DSS) for solving MEX-MDP [5]. Our current work is still focused on the same path with the project MEXAR2 which integrates the algorithms described in this paper in a larger algorithmic framework within a DSS[1]. Within the MEXAR2 project the main idea is to integrate human strategic capabilities and automatic problem solving algorithms to find solutions with *the right compromise* among different and contrasting goals under the full control of the mission planners.

[1]For further details please see `http://mexar.istc.cnr.it`.

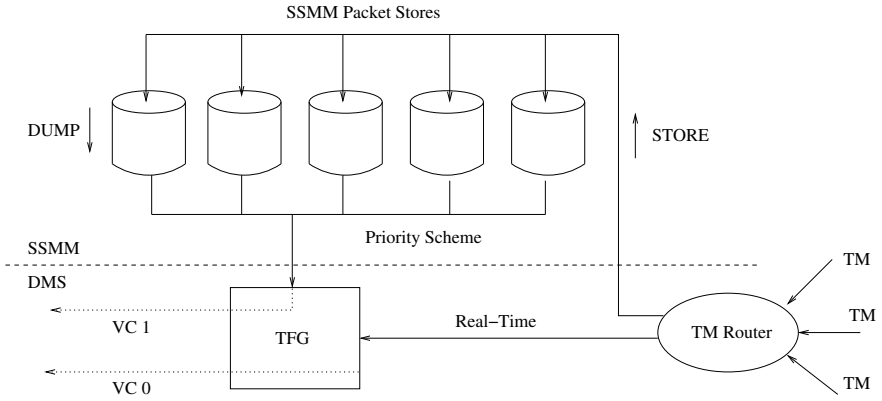


Fig. 1. On-board telemetry flow

In line with this approach, this work proposes a heuristic method to solve MEX-MDP which while it cannot guarantee an optimal solution, yet it can be a very *flexible* solving approach for MEX-MDP. In fact though we could have formulated the whole MEX-MDP as a monolithic multi-criteria mixed-integer programming (MIP) problem, we preferred a heuristic solution, integrated in a larger DSS framework, where a mission planner can drive the search for a dump plan by integrating his knowledge and experience and to build a solution which suits the mission needs.

III. THE MEMORY DUMPING PROBLEM

In a deep-space mission like MARS-EXPRESS data transmission to Earth represents a fundamental aspect. In this domain, a space-probe continuously produces a large amount of data resulting from the activities of its payloads and from on-board device monitoring and verification tasks (the so-called *housekeeping* data). All these data should be transferred to Earth during bounded downlink sessions. Moreover, in the case of MARS-EXPRESS a single pointing system is present. This implies that, during regular operations, the space-probe either points to Mars, to performs payload operations, or points to Earth, to download the produced data. As a consequence, on-board data generally require to be first stored in a Solid State Mass Memory (SSMM) and then transferred to Earth. Therefore, the main problem to be solved consists in synthesizing sequences of spacecraft operations (*dump plans*) that are necessary to deliver the content of the on-board memory during the available downlink windows. This allows to save upcoming pieces of information without losing previously stored data and to optimize given objective functions [4].

The process of collecting data from a remote satellite like Mars-Express is part of a larger planning process involving different stages and the interaction among several teams (Science Working Teams, Mission Analysis Team, Flight Dynamics and Flight Control Teams). In addition, we can identify three phases for planning: long, medium and short term planning, which respectively have durations of six months, one months and one week. Along these phases the planning activities of the satellite are incrementally refined to a granularity of one week. Hence, since some activities cannot be predicted one week in advance (e.g., dump commands for spacecraft operation

requests or flight dynamic request), a *very short-term plan* is generated each one or two days, which includes the *latest* commands. This plan generally covers a period of two days and a dump plan is part of the set of commands uplinked to the satellite.

In the following sections we describe our approach for solving the MEX-MDP problem (part of the very last phase of the whole planning process). The MARS-EXPRESS domain contains many kinds of constraints. For instance, for the communication channel we have different transmission rates according to different periods. Additional constraints arise from the specific use of the on-board memory: in fact, this memory is subdivided into different memory banks (or packet stores) each with finite capacity; each piece of information can be stored in a set of one or more packet stores. Moreover, data are stored in a sequential way while the packet stores are managed cyclically. As a consequence, in case the memory is full, precious data might be overwritten (and then lost) by the newly incoming data.

A. Basic Domain Entities

The Mars Express Memory Dumping Problem (MEX-MDP) has been initially formalized in a previous study conducted by our group for the European Space Agency (see [3]). In the rest of this section we describe the main features which characterize a MEX-MDP instance. Fig. 1 shows a sketch of the MARS-EXPRESS modules that are relevant to MEX-MDP. It shows the different telemetry (TM) data produced on-board and then stored in the *Solid State Mass Memory (SSMM)* that is composed of different packet stores. Stored data are downloaded with different dumps to Earth. The basic entities that are relevant to the MEX-MDP domain can be subdivided into *resources* and *activities*: resources represent domain subsystems able to give services, whereas activities model tasks to be executed using resources over time.

Resources: MEX-MDP requires to model two different types of resources:

- *Solid State Mass Memory (SSMM)*. The SSMM is used to store both science and housekeeping data. The SSMM is subdivided into a set of devices named *packets store*, $\{pk_1, pk_2, \ldots, pk_n\}$, each one with a fixed capacity, c_i. Each packet store can be seen as a file of given maximum size and cyclically managed, that is, previous pieces of information will be overwritten, and then lost, if the amount of stored data overflows the packet store capacity. In particular, for each packet store pk_i it is possible to define a time function $use_i(t)$ that represents the amount of data memorized (that is, the difference between the stored and the dumped data) in the packet store pk_i at the instant t. Given $use_i(t)$ it is possible to define for each packet store pk_i the following conservative constraint:

$$0 \leq use_i(t) \leq c_i \tag{1}$$

- *Communication Channels*. These represent the downlink connections to Earth for transmitting data. These resources are characterized by a set of separated communication windows $\mathcal{CW} = \{cw_1, cw_2, \ldots, cw_{nw}\}$ that identify intervals of time in which downlink connections can be established. Each element w_j is a 3-tuple

$\langle r_j, s_j, e_j \rangle$, where r_j is the available data rate during the time window w_j and s_j and e_j are respectively the start and the end-time of this window.

Activities: these describe *how* resources can be used. Each activity a_i is characterized by a fixed duration d_i and two variables s_i and e_i which respectively represent its start-time and its end-time. Two basic types of activity are relevant to MEX-MDP: store operations st_i and memory dumps md_i.

- *Store Operation*. Each store operation, st_i, "instantaneously" stores, at its end-time e_i, an amount of data q_i in a defined packet store, pk_i.
- *Memory Dump*. Through a memory dump, $md_i = \langle pk_i, s_i, e_i, q_j \rangle$, an amount q_i of stored data is transmitted from the packet store pk_i to the ground station, during the interval $[s_i, e_i]$.

In the MEX-MDP domain two different kinds of data can be modeled by using store operations st_i: the *Payload Operation Request* (POR) and the housekeeping activities. The latter ones produce a continuous stream of data at a given constant rate, called *Continuous Data Stream* (CDS).

A *Payload Operation Request* is a scientific observation which generates a set of data distributed over the available packet stores. According to this model, for each por_i, the produced data are decomposed in a set of different *store* operations such that, $por_i = \{st_{ij}\}$. All of these store operations have the same durations and the same start-times, that is, in our model the different data are stored in the different packets store at the same time.

On the other hand, a *Continuous Data Stream* models an on-board process which works in "background" with respect to the scientific activities. This process generates a flow of data with constant rate which has to be stored in the SSMM. Examples of such data streams are the housekeeping data collected on a regular basis to control the behavior of the on-board sub-systems.

The two data sources exhibit different characteristics: in fact, a POR is a time bounded activity, which stores data at its end-time, whereas a CDS is a continuous data flow over the domain horizon. However, we choose to model a CDS as a periodic sequence of store operations. In particular, given a CDS with a data flat rate r, we define a period T_{cds}, such that, for each instant of time $t_j = j \cdot T_{cds}$ ($j = 0, 1, 2, \dots$) an activity st_{ij} stores an amount of data equal to $r \cdot T_{cds}$. In the particular case of $t_0 = 0$ we suppose the amount of stored date is zero. Hence, we can consider as input data for the problem just an equivalent set of store operations.

B. Problem Definition

Given the domain entities of a MEX-MDP instance, in this section we introduce the definition of solution, and *robust* solution for this problem. It is worth noting that our definitions refers to a domain abstraction that allows to bring out the relevant aspects of the problem. In particular, we consider two levels of abstraction for the problem domain.

- In a first level, *Data Dump level*, it is assessed the amount of data to dump from each packet store for each time window.
- In a second level, *Packet level*, the final dump commands are generated from the first level results.

It is worth remarking that the second step can be automatically accomplished once a solution for the Data Dump level is achieved (see Algorithm 1). For this reason along our research work we mainly have focused the attention exclusively on producing solutions for the Data Dump level.

The problem *decomposition* described above is motivated by the complexity of the problem. In fact, this abstraction allows us to focus on the *dominant* aspects of the problem, i.e. data quantities, packets store capacities, and dump capability over the communication links, without considering the problem of generating dump commands.

A MEX-MDP instance is then composed of a set of scientific observations, $\mathcal{POR} = \{por_1, por_2, \ldots, por_{np}\}$ and a set of housekeeping productions, $\mathcal{CDS} = \{cds_1, cds_2, \ldots, cds_{nc}\}$, which are both modeled with a set of store operations, and a time horizon $\mathcal{H} = [0, H]$. A *solution* to a MEX-MDP instance is a set of dumping operations $S = \{md_1, md_2, \ldots, md_{nd}\}$ *such that*:

- At each instant of $t \in \mathcal{H}$, the amount of data stored in each packet store pk_i most not exceed the packet store capacity c_i, i.e., overwriting is not allowed. See (1).
- The whole set of data must be "available" on ground within the considered temporal horizon $\mathcal{H} = [0, H]$, except an amount of residual data for each packet store pk_i less or equal to the capacity c_i at $t = H$ which cannot be dumped within $\mathcal{H}$.
- Each dump activity, md_i, is executed within an assigned time window w_j which has a constant data rate r_j. Moreover, dump operations cannot mutually overlap.

Robustness and Uncertainty. Though a solution should satisfy all the imposed constraints, a further goal is to find *high quality* solutions with respect to *robustness* properties. Informally, *a high quality plan delivers all the stored data and is able to "adsorb" external modifications that might arise in a dynamic execution environment.*

In fact, in the problem our work focuses upon, sources of uncertainty stem mainly from the unpredictability of the scientific observations' outcome. For instance, the data volume produced by the High Resolution Stereo Camera (HRSC) - one of the most memory consuming payload - depends on the target and on the context. Thus the impossibility to have an exact estimation of the memory usage of each request leads to producing brittle solutions.

In particular, in the case of the MEX-MDP problem our aim is to control the level of memory use in order to avoid possible loss of data due to overwriting. One possibility for overwriting can occur when a greater than expected volume of data has to be stored and there is not enough space in the packet store. For this reason we define a robust solution a solution in which a specified amount of space of each packet store is preserved in order to safeguard against overwriting. In other words, solution's robustness is related to the concept of *distance to the overwriting state*. Hence, we consider the peaks of data in a packet store close to its maximum capacity as sources of schedule's brittleness.

A possible way to increase solution's robustness is to *flat* these peaks by finding a different distribution of the dumping operations within the horizon $[0, H]$. Let $use_i^{(max)}$ be the maximum value over the horizon $[0, H]$ of $use_i(t)$. We define the packet store utilization $\alpha_i = use_i^{(max)}/c_i$ as the ratio between the maximum level of data in the packet store pk_i and its capacity c_i. The robustness of a solution S is defined as:

$$r(S) = \max_{i=1,\ldots,n}\{\alpha_i\} = \max_{i=1,\ldots,n}\{use_i^{(max)}/c_i\} \tag{2}$$

that is the maximum value of packet store utilization. A solution S is *optimal* when $r(S)$ is minimal.

IV. A MAX-FLOW APPROACH FOR MEX-MDP

In this section we recall a previous formalization of the MEX-MDP problem [4], and then we show as this formalization matches with a flow network model. This allows in the next section to define a solving procedure for the MEX-MDP problem based on a Max-Flow solver.

A. MEX-MDP *formalization*

The formalization is based on a partition of the temporal horizon $\mathcal{H} = [0, H]$ into a set of contiguous windows $W = \{w_1 = [t_0, t_1] \mid t_0 = 0\} \cup \{w_j = (t_{j-1}, t_j] \mid j = 2 \ldots m, t_i \in \mathcal{H}\}$, such that $\cup_{j=1}^{m} w_j = \mathcal{H}$. The partition is realized upon consideration of significant events. Such events are assumed to be the start and the end of the temporal horizon, the time instants where a memory reservation on a packet store is performed, and the time instants where a change on the channel data rate is operated. It is assumed that such significant events take place at the windows' edges. In this way inside the windows w_j it is possible only to dump data and no store is executed. The key point of the formalization is represented by the decision variables:

$$\delta_{ij} \qquad i = 1, \ldots n,\ j = 1, \ldots m, \tag{3}$$

each defined in the integer domain $[0, c_i]$. These represent the amount of data to dump from the packet store pk_i within a window w_j. To formally represent the domain constraints, for each packet store pk_i $(i = 1, \ldots, n)$ and for each time window w_j $(j = 1, \ldots, m)$ some additional definitions are needed:

- d_{ij}, amount of data memorized in the packet store pk_i at t_j. Where the variables $d_{i0} \leq c_i$ represent the initial data level in the packet store pk_i;
- l_{ij}, maximal level (amount of data stored) allowed at t_j for the packet store pk_i, $l_{ij} \in [0, c_i]$;
- b_j, maximal dumping capacity available in w_j;

We introduce two classes of constraints on the set of decision variables δ_{ij}. A first constraint captures the fact that for each window w_j the difference between the amount of generated data and the amount of dumped data cannot exceed the maximal imposed level in the window l_{ij} (*overwriting*). Additionally, the dumped data cannot exceed the generated data (*overdumping*). Thus, the *conservative constraint* (1) is redefined into the following inequalities:

$$\sum_{k=0}^{j} d_{ik} - \sum_{k=1}^{j} \delta_{ik} \leq l_{ij} \tag{4}$$

$$\sum_{k=0}^{j-1} d_{ik} - \sum_{k=1}^{j} \delta_{ik} \geq 0 \tag{5}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. The need of these two inequalities is justified by the example in Fig. 2, where we represent the changing over time of the stored data volume for a generic packet store pk_i with respect to a window w_j. In particular, the *store* operations are "impulsive actions" performed at the extremes (t_{j-1} and t_j) of w_j, whereas the dumping operations are performed *within* the window w_j. As a consequence, the amount of data d_{ij}, stored at t_j, cannot be dumped in the window w_j, because the data are not available during w_j. Hence, for each window w_j, we have to check constraint (5) just before t_j (at t_j^-), and constraint (4) just after t_j (at t_j^+). These constraints are necessary to avoid, respectively, to dump more data than available and to store more data than allowed by the value of l_{ij}. A second class of constraints takes into account the dumping capacity imposed by

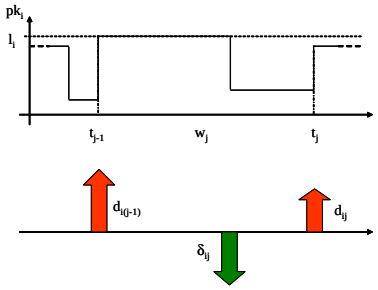


Fig. 2. The conservative constraints (4) and (5) have to be checked respectively at t_j^+ and t_j^-.

the communication channel. Let b_j be the maximum amount of data that can be dumped during w_j, we have that:

$$0 \leq \sum_{i=1}^{n} \delta_{ij} \leq b_j \quad j = 1, \ldots, m \tag{6}$$

these *downlink constraints* state that for each window w_j, is not possible to dump more data than the maximum dumping capacity b_j.

B. Flow Networks and the Max-Flow Problem

Before the introduction of the flow network model for MEX-MDP, in the following we briefly review the theory behind the Max-Flow problem [6]. A flow network $G(V, E)$ is a direct graph where V is a set of vertices and E is a set of edges (u, v) with nonnegative capacity $c(u, v) \geq 0$. The flow network has two special vertices: a source s and sink t. A flow in G is a integer-valued function $f : V \times V \rightarrow \mathbb{Z}$ (we consider only integer-valued flows) that satisfies the following three properties:

- capacity constraint: for all $u, v \in V$,

$$f(u, v) \leq c(u, v)$$

- skew symmetry: for all $u, v \in V$,

$$f(u, v) = -f(v, u)$$

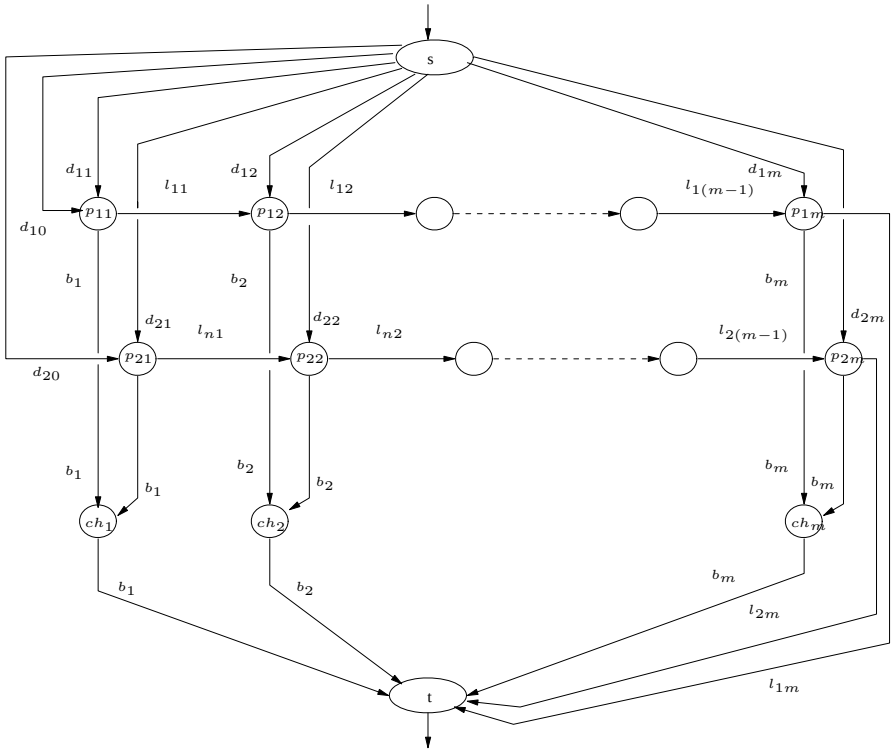


Fig. 3. A flow network for MEX-MDP.

- flow conservation: for all $u \in V \cup \{s, t\}$,

$$\sum_{v \in V} f(u, v) = 0$$

.

The quantity $f(u, v)$ can be positive or negative, and it represents the *net flow* from vertex u to vertex v. The value of a flow f into the graph G, is defined as

$$f = \sum_{v \in V} f(s, v),$$

that is the total flow out of the source. In the Max-Flow problem given a flow network G, the goal is to find a flow of maximum value from source to sink.

C. A Flow Network model for MEX-MDP

In this section we introduce a flow network to model a MEX-MDP instance. Fig. 3 shows an example of flow network in the case of two packets store, however this example can be easily generalized to the case of n packets stores. There are four types of nodes: *source*, *sink*, packet-store nodes p_{ij} (as explained below such nodes are actually *macro-nodes*) and channel nodes ch_j.

The packet store node p_{ij} is composed of two nodes (see Fig. 4) to represent the two conservative constraints (4) and (5). The first node $p_{ij}^{(1)}$ represents the *overdumping* constraint (5), such that, within the window $w_j = [t_{j-1}, t_j]$ it is not possible to dump more data than the amount available at t_{j-1}. On the other hand, the node $p_{ij}^{(2)}$, represents the *overwriting* constraint (4), such that, the amount of residual data in the packet store after the dumping operation over the window w_j, added to the amount of data stored at t_j (d_{ij}), is less or equal to the allowed capacity of the packet store l_{ij}. In particular, on the node $p_{ij}^{(1)}$ there are the following three flows: $f(p_{i(j-1)}, p_{ij})$, $f(p_{ij}, ch_j)$, and $f(p_{ij}^{(1)}, p_{ij}^{(2)})$. The flow $f(p_{i(j-1)}, p_{ij})$ represents the residual amount of data on pk_i at t_{j-1}, that is, $\sum_{k=0}^{j-1} d_{ik} - \sum_{k=1}^{j-1} \delta_{ik}.f(p_{ij}, ch_j)$ represents the amount of data dumped from pk_i during the time window w_j, that is $\delta_{ij} = f(p_{ij}, ch_j)$. This edge is labeled with $c(p_{ij}, ch_j) = b_j$ to hold the constraint $\delta_{ij} \leq b_j$, that is, *it is not possible to dump more data than the channel capacity* b_j. Finally the flow $f(p_{ij}^{(1)}, p_{ij}^{(2)})$ represents the amount of residual data which remains in the packet store after the dumping over the window w_j.

We remark that we split the node p_{ij} for the same motivation for which the inequality (1) has been split into the inequalities (4) and (5). In fact, if we consider the flow balance on the node $p_{ij}^{(1)}$ we have that:

$$f(p_{i(j-1)}, p_{ij}) - f(p_{ij}, ch_j) = f(p_{ij}^{(1)}, p_{ij}^{(2)})$$

that can be rewritten as:

$$\sum_{k=0}^{j-1} d_{ik} - \sum_{k=1}^{j-1} \delta_{ik} - \delta_{ij} = f(p_{ij}^{(1)}, p_{ij}^{(2)}) \geq 0$$

which coincides with the inequality (5). On the other hand, on the node $p_{ij}^{(2)}$ there are the following flows: $f(p_{ij}^{(1)}, p_{ij}^{(2)})$, $f(s, p_{ij})$, and $f(p_{ij}, p_{i(j+1)})$, where the latter represents the residual amount of data $\sum_{k=0}^{j} d_{ik} - \sum_{k=1}^{j} \delta_{ik}$, and the flow $f(s, p_{ij})$ concerns the amount of data stored at the window w_j, which can be at most equal to d_{ij}. Considering the flow balance on $p_{ij}^{(2)}$ we have that:

$$f(p_{ij}^{(1)}, p_{ij}^{(2)}) + f(s, p_{ij}) = f(p_{ij}, p_{i(j+1)})$$

that can be rewritten as:

$$\sum_{k=0}^{j-1} d_{ik} + d_{ij} - \sum_{k=1}^{j} \delta_{ik} = f(p_{ij}, p_{i(j+1)}) \leq l_{ij}$$

which coincides with the inequality (4).

To represent the initial value of each packet store the arcs $(s, p_{i0}^{(1)})$ are labeled with the value d_{i0} whilst the arcs $(p_{im}^{(2)}, t)$ are used to represent the overall *residual* amount of data on the packets store pk_i at $t = H$. To synthesize solutions which dump all the data on the ground these arcs are labeled with $l_{im} = 0$.

Finally the flow through each channel node ch_j represents the *downlink* constraint over the time window w_j,

$$\sum_{i=1}^{m} f(p_{ij}, ch_j) = \sum_{i=1}^{m} \delta_{ij} = f(ch_j, t) \leq b_j$$

that is, *the sum of data dumped from each packet store over the window* w_j *is less or equal the channel capacity* b_j.

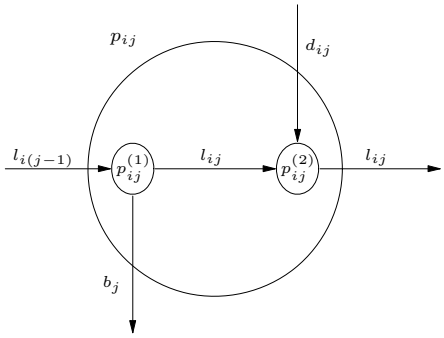


Fig. 4. The internal structure of a macro-node p_{ij}.

To conclude, a flow assignment to any edges is computed by applying a Max-Flow algorithm to the flow network in Fig. 3. The result represents the maximum value of data which can be conveyed through the communication channel. Thus a MEX-MDP will admit a solution **if and only if**:

$$f(s, p_{ij}) = c(s, p_{ij}) = d_{ij}$$

for each $i = 1, \ldots, n$ and $j = 1, \ldots, m$. In other words, the formula above states that the maximum flow through the network equates the whole data set stored in the different packets store.

Based on the model described above, in the next section two methods for approaching a MEX-MDP instance are introduced: the first computes a solution, while the second aims at refining an initial solution for the sake of robustness.

V. SOLVING METHODS

As mentioned above, to find a solution to a MEX-MDP problem is sufficient to apply a Max-Flow algorithm to the associated flow network (see Fig. 3), next the set of dump commands can be obtained with a simple algorithm. A solution of a MEX-MDP instance, mdp, exists if and only if the flow through each arc (s, p_{ij}) equates its capacity value, that is, $f(s, p_{ij}) = c(s, p_{ij})$. In such a case we have that the set of values $\delta_{ij} = f(p_{ij}, ch_j)$ is a solution of mdp. There are different polynomial algorithms to solve the Max-Flow problem. Our current implementation is based on the Edmond-Karp version of the Ford-Fulkerson method. The Edmond-Karp algorithm runs in $O(|V||E|^2)$ time, where $|V|$ and $|E|$ are respectively the number of nodes and the number of arcs in the flow network. Being in our case $|V| = O(m\,n)$ and $|E| = O(m\,n)$, our current implementation runs in $O(m^3 n^3)$. An improvement in the sense of CPU time can be achieved using more sophisticated max flow methods like the preflow-flush one [7]. The reader can find an essential survey on this issue in [6, chapter 26].

A. Finding a downlink schedule

We now describe how to build a downlink schedule. As mentioned above, a downlink schedule $S = \{md_1, md_2, \ldots, md_{nd}\}$ can be directly deducted on the basis of the values δ_{ij}.

Algorithm 1: Make a downlink schedule

input : a solution δ_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$

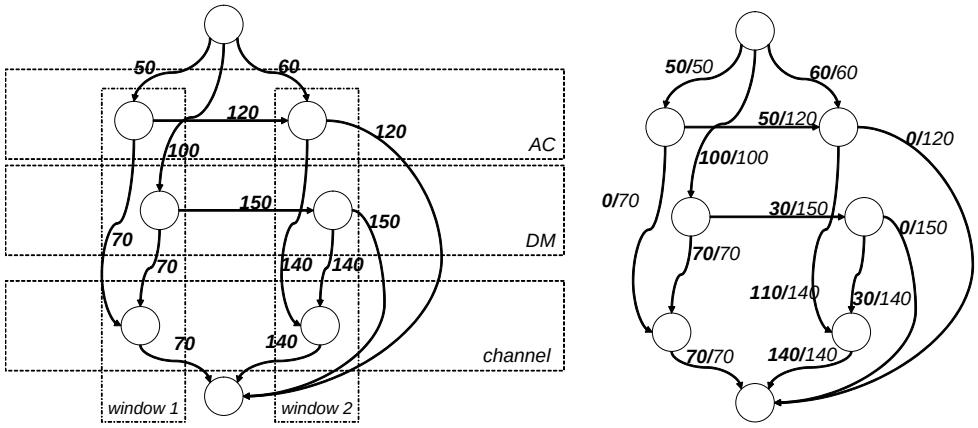
output: a schedule $S = \{md_1, md_2, \ldots, md_{nd}\}$

1 **begin**
2 $S \leftarrow \emptyset$
3 $c \leftarrow 1$
4 $t \leftarrow t_0$
5 **foreach** *(window* w_j, $j = 1, \ldots, m$*)* **do**
6 **while** $\exists$ *a packet store* pk_i*:* $\delta_{ij} > 0$ **do**
7 Select a packet store pk_c: $\delta_{cj} > 0$
8 $s_c \leftarrow t$
9 $e_c \leftarrow s_c + \delta_{cj}/r_j$
10 $md_c = \langle pk_c, s_c, e_c, \delta_{cj} \rangle$
11 $S = S \cup \{md_c\}$
12 $t \leftarrow e_c$
13 $c \leftarrow c + 1$
14 $t \leftarrow t_j$
15 **return** S
16 **end**

Algorithm 1 takes as an input a solution δ_{ij} ($i = 1, \ldots, n$, $j = 1, \ldots, m$) and finds a downlink scheduling S by iterating on the set of downlink windows $w_j = [t_{j-1}, t_j]$ ($j = 1 \ldots m$) . For each window w_j, the algorithm selects a packet store pk_c until all the packet stores with variables $\delta_{cj} > 0$ are all selected. Within the *While* loop the packet store selection (Step 7) is performed on the basis of an heuristic criterion. Many criteria are possible, for example the *shortest dump first* heuristic, which selects first the packet store with the smallest value $\delta_{cj}0$. A memory dump operation $md_c = \langle pk_c, s_c, e_c, \delta_{cj} \rangle$ from a packet store pk_c is obtained by adding its start time s_c, end time e_c. The start time is calculated by means of a variable t (current time) which is incremented by the duration of the current added memory dump (within the while loop) or set to the start of the next dump window w_j when all the packet store within w_j are considered.

Example 1: To better explain our flow model and algorithm to convert a max-flow solution into a downlink schedule, let us introduce the following example. Two payloads are part of the satellite, with ids AC and DM, such that, each one stores its own data in an exclusive packet store which has the same id of the related payload. We have three store operations on the packet stores AC and DM:

- por_1 on AC which produces 50 Mb at t= 11:55:11;



(a) Flow-Network model (b) Max-Flow solution

Fig. 5. Flow network for Example 1

- por_2 on DM which produces 100 Mb at t= 12:13:37;
- por_3 on AC which produces 60 Mb at t= 15:33:12.

In the period taken into account we have two transmission windows:

- $window_1$, from 12:20:12 to 12:43:32, rate= 50 Kbps;
- $window_2$, from 17:25:50 to 18:59:10, rate= 25 Kbps.

Considering the duration (respectively 00:23:20 and 01:33:20) and the transmission rate of each window we have a dump capacity of 70Mb and 140Mb respectively. Furthermore, the two packets store, AC and DM, have a maximum capacity equal to 120Mb and 150Mb respectively.

Figure 5(a) shows a flow-network representation for the problem introduced so far. We emphasize with labeled dashed rectangles the nodes of the network which belong to each packet store, to the transmission channel and to each time window. As described above there are five types of edges:

- from the *source* to a “packet store” node (we omit the edge if the capacity/the dump value is zero) to represent the amount of data to download;
- between nodes of the same packet store to represent the packet store availability;
- from a “channel” node and the *sink* to represent the channel capacity for each time window (in this case we have 70Mb and 140Mb);
- from a “packet store” to a “channel” node to represent the data dumped from each packet store within each time window.
- from a “packet store” node to the sink (of course these are labeled with the packet store capacity value) to represent the amount of residual data for each packet store at the end of the planning horizon.

Figure 5(b) presents a max-flow solution to the network Fig. 5(a). The label of each edge denotes respectively the flow through the edge and its capacity. We can observe on this solution the following points: – all the dump

data are downloaded, in fact there is no flow through the edges between the "packet store" nodes and the sink and there is no possibility to increase the flow through the edges between the source and a "packet store" node; – there are three dump operations from both the packets stores at each time window (see edges between "packet store" and "channel" nodes).

It is worth noting that the Max-Flow approach allows to denote the number and the size of dump operation for each time window. Given this solution it is straightforward to compute the exact dump commands by Algorithm 1. For instance in this case we have the following list:

start	stop	PkS	Dumped[Mb]
12:20:12	12:43:32	DM	70
17:25:50	17:45:50	DM	30
17:45:50	18:59:10	AC	110

Each item represents a different dump command and specifies the start and end-time of the dump, the packet store involved and the amount of dumped data.

We conclude by remarking that given the results of the Max-Flow algorithm the list above is not unique. In this example, we use the *shortest dump first* heuristic, that is given a set of planned dumps within a specific transmission window, we schedule them in increasing order w.r.t. the amount of dumped data. For this reason during the second transmission window we first schedule the dump from the packet store DM (30 Mb instead of 110 Mb).

B. Iterative Leveling: Improving Robustness

In this section we present the iterative algorithm used to improve the robustness of an input solution. We recall that in this domain we consider a solution as robust if the level of data over time of each packet store has no *peaks* close to its maximal capacity, so that there is always available memory for unexpectedly large amounts of data. Hence, roughly speaking, we can remove *dangerous* peaks of data with a *leveling* procedure which distributes the "exceeding" data over the problem horizon. In particular, we propose an heuristic algorithm for improving robustness which iteratively applies the three-steps cycle *solution analysis/problem-update/construction*. Our approach somehow resembles the concept of "feedback" widely used in Control Theory. Furthermore, a similar idea has been proposed in the work [8] for the optimization of the makespan of scheduling problems.

The iterative method is presented with Algorithm 2 (*Iterative-Leveling*). This takes in input a MEX-MDP instance mdp, and a parameter $\epsilon \in (0, 1)$. The algorithm starts by finding an initial solution, represented in compact way by $\delta = \{\delta_{ij} : i = 1, \ldots, n, j = 1, \ldots, m\}$ (Step 2). If a solution is found, $\delta \neq \emptyset$, the algorithm proceeds by initializing all the elements of the vector $flatten[]$ to TRUE, where $flatten[i] = \text{TRUE}$ means that the maximum usage of the packet store pk_i can be potentially lowered again.

The **while** loop (Steps 6-16) represents the core of the algorithm. In this loop the following three steps are repeated:

1) analyze the current solution and select a *critical* packet store pk_k, such that the percentage usage value α_k is maximum (Step 7);

Algorithm 2: Iterative-Leveling

input : a MEX-MDP instance, mdp, and a parameter $\epsilon \in (0, 1)$

output: a MEX-MDP solution δ

```
1  begin
2      δ ← SolveByMaxFlow(mdp)
3      if δ ≠ ∅ then
4          for j = 1 to m do
5              flatten[j] ← TRUE
6          while ∃i|flatten[i] =TRUE do
7              pk_k ← SelectPacketStore()
8              for j = 1 to m do
9                  l_kj ← α_k(1 − ε)l_kj
10             δ′ ← SolveByMaxFlow(mdp)
11             if δ′ = ∅ then
12                 flatten[k] ← FALSE
13                 for j = 1 to m do
14                     l_kj ← l_kj / (α_k(1−ε))
15             else
16                 δ ← δ′
17     return δ
18 end
```

2) for any time window w_j, the maximum level constraint l_{kj} of the selected packet store pk_k is reduced to the value $\alpha_k(1-\epsilon)l_{kj}$ (Steps 8-9). In this way the maximum percentage usage is forced to be less than α_k;
3) solve the modified problem (Step 10). If this does not admit a solution then the "modified" packet store pk_i, is labeled as not improvable $flatten[i] = \text{FALSE}$ and the previous consistent situation is reloaded (Steps 13-14). Otherwise the current best solution is updated (Step 16).

The aim of the three steps is to iteratively flatten the current critical packet store. These steps will be repeated until there is at least one packet store which admit an improvement.

Example 2: To better explain the robustness concept as well as the iterative leveling algorithm, we consider again the problem introduced in Example 1.

Figure 6 shows the profiles usage of both the packets store with respect the computed solution (see Fig. 5(b)).

Even though the two profiles are consistent with respect to the packet store capacity, we notice that for the AC

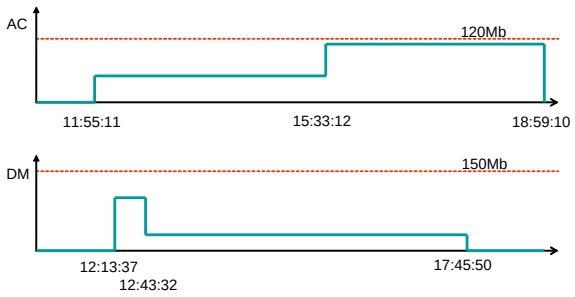


Fig. 6. Packet Store Profiles usage w.r.t. the solution in Fig. 5(b)

packet store we have a critical situation. At time t= 15:33:12 we have a peak of memory usage equal to more than 91% (we have an amount of stored data equal to 110Mb against a capacity of 120Mb). Regarding DM we instead have a maximum use equal to $66,7\%$.

The criticality of this situation stems from the unpredictability of the operation outcome. In fact if the second operation (at time t=15:33:12) produced more data, for instance 75Mb, we would encounter a data loss. For this reason we have shown above that a possible approach may consist in reducing the capacity considered during the Max-Flow algorithm, in order to anticipate some dumps from critical packets store.

In this case we can "robustify" the solution by reducing the resource capacity considered for the packet store AC. For instance, Fig. 7 shows the result of our algorithm considering for AC a capacity of 100Mb instead of 120Mb by means of an input parameter $\epsilon = 1/6$ (Note that the edge between the two "packet store" nodes associated with AC is now labeled with the value 100). To avoid an inconsistent situation (w.r.t. the new constraints), a first dump (50Mb) from AC is scheduled during the first transmission window. This allows to eliminate the previous peak. Of course the amount of data dumped from DM during the same interval is reduced (in Fig. 7 the new flows value are underlined). Given this solution a possible sequence of dump commands is the following:

start	stop	PkS	Dumped[Mb]
12:20:12	12:26:52	DM	20
12:26:52	12:43:32	AC	50
17:25:50	18:05:50	AC	60
18:05:50	18:59:10	DM	80

Figure 8 presents the new profiles for the new solution. We have now that the maximum usage for AC is 50% while for DM is still $66,7\%$. Therefore the new solution presents more robust characteristics than the one introduced in the Example 1.

VI. Experimental Evaluation

This section is dedicated to discuss the results of an empirical evaluation of the methods described above. These have been evaluated using the benchmark sets defined during the study conducted for the European Space Agency

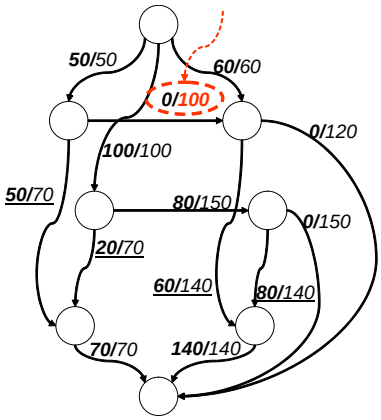


Fig. 7. Alternative, more robust, solution.

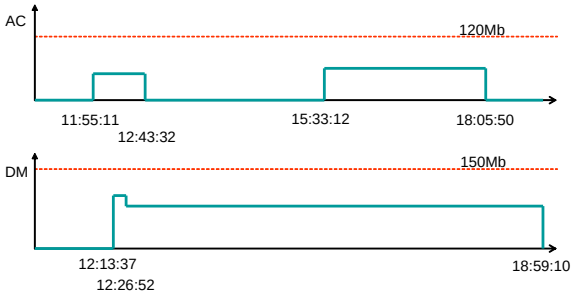


Fig. 8. Packet Store Profiles usage w.r.t. the solution in Fig. 7

[3]. In particular, in this section we present the results for one of these benchmark sets[2], B5. This benchmark is composed of 9 problems and has been generated on purpose in order to be critical with respect to, on one hand, the competition among the packet stores for the same channel bandwidth, and, on the other hand with respect to the limited capacity of the packet stores relatively to the amount of generated data. In particular, these problems are generated with regard to the following setting for the domain parameters: 1 science housekeeping packet store, 11 science packet stores, 8 payloads and a channel data rate of 228 Kbps.

Figure 9 shows the results with respect to the solution's robustness (see Section III-B): the application of the *Iterative-Leveling* algorithm increases the quality of the final solution. In particular, the graph labeled with $INIT$ represents the robustness of a solution generated with one run of the solving algorithm based on the Max-Flow reduction, while the curve labeled with *LEV* represents the robustness values after the application of the *Iterative-Leveling* algorithm described in Section V with $\epsilon = 0.02$. Figure 9 shows how for some problems the robustness is improved of 25%, that is, the maximum utilization of a packet store (2) is lowered from 100% to 75%.

In addition, a further analysis can be done considering the average maximum utilization of the packet stores, that

[2]The benchmark sets are available at the address: `http://mexar.istc.cnr.it`.

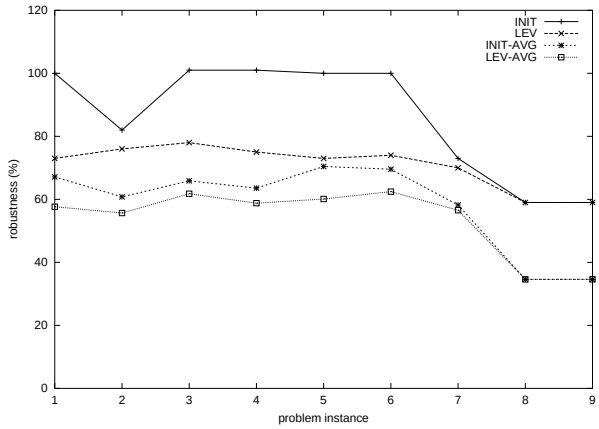


Fig. 9. Performance on benchmark B5.

is, the average of the values $use_i^{(max)}/c_i$ over the set of the actually used packet stores. The two curves, *INIT-AVG* and *LEV-AVG* in Fig. 9 represent this value respectively before and after the application of the *Iterative-Leveling* algorithm. Clearly, the main effect of the leveling algorithm is to create a different distribution of the dumping operations over the horizon in order to remove *dangerous* data peaks. Regarding the CPU-time, all the algorithms presented in this paper are implemented in Java on an Athlon 1800 Mhz machine, and the average CPU-times are respectively 0.8 seconds to generate an initial solution and 21.8 seconds to improve its robustness.

VII. Mission Operative Environment and Robustness

The algorithms described in this paper are part of the software system MEXAR2, a Decision Support System (DSS) developed within a project work supported by ESA and targeted for solving the MEX-MDP. In the MEXAR2 project the main idea is to integrate human strategic capabilities and automatic problem solving algorithms to find solutions with the *right compromise* among different and contrasting goals, under the full control of the mission planners. As introduced above, in this work we refer to an abstract *core* model of MEX-MDP, which allows us to introduce the Max-Flow model of the problem and to recognize one of the main source of brittleness for a downlink schedule, i.e. peaks of data volumes stored in the on-board memory. However, in the realm of the mission operative environment, a mission planner have to take into account a set of additional constraints for MEX-MDP with respect to those modeled in Section III-B. Under this additional set of constraints, the MEX-MDP problem could be formulated as a monolithic multi-criteria mixed integer programming (MIP) problem, but we prefered a heuristic approach for its solution, like the Max-Flow based one. In fact, this approach allows a finer control of the search for dump plans by adopting different heuristics for the selection of augmenting paths within the Max-Flow algorithm, such that this flexibility can be exploited for a smoother integration of the mission planner choices in the search process and to cope with the following further additional constraints for MEX-MDP:

- in general, a MEX-MDP admits many solutions, however from a practical point of view, the only useful ones

are the solutions where each dump operation starts as soon as possible. In other words, useful solutions are the ones where the variables δ_{ij} greater than zero are *shifted* toward time origin. These kind of solutions can be easily obtained with a max-flow algorithm by generating the augmenting path through the channel nodes in increasing order of indexes.

- When the problem is *over-constrained*, that is, no solution exists and part of the data is lost (this is an extreme situation in the MARS-EXPRESS domain, but possible). The solving procedure must return in every case a solution, which *sacrifices* some data. In this case the solution provided by the max-flow is still useful, even if, some data is lost. Also, in this case it is possible for a user to select which data to sacrifice by driving the search of augmenting paths within the max-flow algorithm.
- Dump commands have a minimal durations (e.g., 30 seconds) and it does not make sense to have commands with durations of few seconds. In general, plan *fragmentation* must be avoided, that is plans containing many commands with small durations interleaving dump commands from several packet stores.
- Packet stores have priorities. These priority values are considered also when some data must be renounced.
- For some packet stores *preemption* is not allowed, hence it is not possible to dump their content over a sequence of dump commands and the operation must be accomplished in one step.

Furthermore, we would like to remind as the solution of MEX-MDP is part of a larger decision process involving several stages of planning (long, medium and short-term planning) the interaction of many working teams. Then, at this level of the decision chain, the role of a DSS is to complete and make executable a set of decisions already taken in an abstract way. For instance, at the end of each mission day, the real volumes of data in the set of packet stores are reported to the mission planner, such that a new downlink schedule can be possibly re-synthesized accordingly. This daily procedure can be seen as a *closed-loop* planning, which reacts to unexpected peaks of data by anticipating the dump operations (this is the main effect of reducing the capacity of a packet stores) in the packet stores which exhibit this kind of criticality. In this situation a robust solution may play a fundamental role in order to avoid plan regeneration and/or data losses.

VIII. CONCLUSIONS

During a project work for the European Space Agency a specific scheduling problem arose: the so called MARS-EXPRESS Memory Dumping Problem or MEX-MDP [3], [4], [5].

In this paper we face this problem in a novel way by a reduction of the MEX-MDP to a Max Flow problem [6]. The algorithm described in this work is currently integrated in a larger algorithmic framework within a Decision Support System (DSS) targeted to deliver high quality solutions to the MEX-MDP where the main goal is to avoid data overwriting, while taking into account other quality measures, like data priorities-based objective functions or the length (number of commands) of a dump plan. Max-Flow reduction can be intuitive considering that a solution to the dumping problem can be seen as a flow from the satellite to the ground, such that the problem has a solution when the maximum flow equates the total amount of data to dump. Given this reduction, a definition of solution's robustness is proposed together with an iterative procedure to improve the robustness of a solution, the underlying

idea being that the lower the memory utilization the higher the ability to face unexpectedly larger amount of data. Experimental data confirm our thesis: we can effectively remove *dangerous* peaks of data and distribute them over the problem horizon by using the *Iterative-Leveling* procedure. We remark that even though the MEX-MDP problem comes from a specific study, its features are quite general and many of the conclusions reported in this paper can be extended to other spacecraft domains which adopt the same model of on-board memory.

ACKNOWLEDGMENTS

The MEX-MDP has been studied in a study conducted for ESA from November 2000 to July 2002 (contract No. 14709/00/D/IM). The Max-Flow approach is currently used in the framework of the project MEXAR2 supported by ESA (contract No. 18893/05/D/HK(SC)).

The authors would like to thank their colleagues Amedeo Cesta and Riccardo Rasconi for their precious advices and suggestions.

REFERENCES

[1] G. Verfaillie and M. Lemaitre, "Selecting and Scheduling Observations for Agile Satellites: Some Lessons from the Constraint Reasoning Community Point of View," in *Principles and Practice of Constraint Programming, 7^{th} International Conference, CP 2001*, ser. Lecture Notes in Computer Science, T. Walsh, Ed., no. 2239. Springer, 2001, pp. 670–684.

[2] E. Bensana, M. Lemaitre, and G. Verfaillie, "Earth Observation Satellite Management," *Constraints: An International Journal*, vol. 4, no. 3, pp. 293–299, 1999.

[3] A. Cesta, A. Oddi, G. Cortellessa, and N. Policella, "Automating the Generation of Spacecraft Downlink Operations in MARS EXPRESS: Analysis, Algorithms and an Interactive Solution Aid," ISTC-CNR [PST], Italian National Research Council, Tech. Rep. MEXAR-TR-02-10 (Project Final Report), July 2002.

[4] A. Oddi, N. Policella, A. Cesta, and G. Cortellessa, "Generating High Quality Schedules for a Spacecraft Memory Downlink Problem," in *Principles and Practice of Constraint Programming, 9^{th} International Conference, CP 2003*, ser. Lecture Notes in Computer Science, F. Rossi, Ed., no. 2833. Kinsale, Ireland: Springer, 29 September - 3 October 2003, pp. 570–584.

[5] G. Cortellessa, A. Cesta, A. Oddi, and N. Policella, "User Interaction with an Automated Solver. The Case of a Mission Planner," *PsychNology Journal*, vol. 2, no. 1, pp. 140–162, 2004.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Second Edition. MIT Press, 2001.

[7] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum flow problem," *Journal of ACM*, vol. 35, no. 4, pp. 921–940, October 1988.

[8] D. Joslin and D. Clements, ""Squeaky Wheel" Optimization," *Journal of Artificial Intelligence Research*, vol. 10, pp. 353–373, 1999.